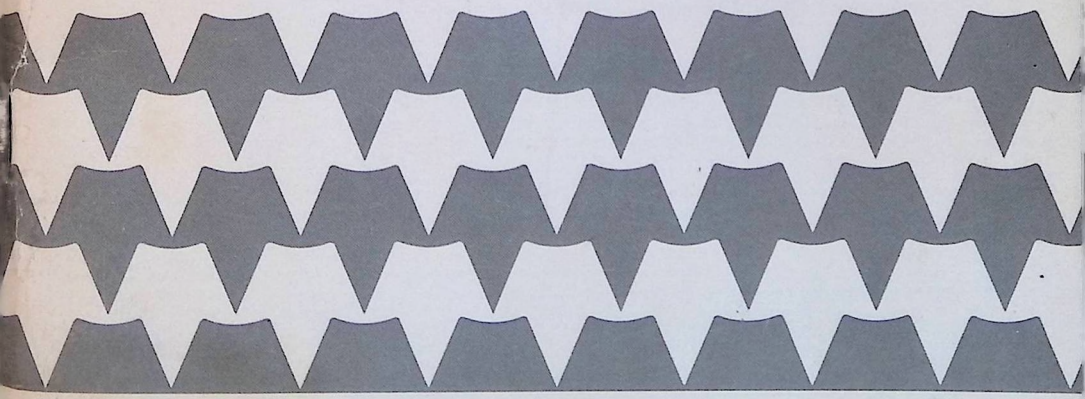
TANDY™
COMPUTER PRODUCTS

Cat. No. 25-1010

Tandy® 1000 Mouse Controller/Calendar Board Owner's Manual



CUSTOM MANUFACTURED IN USA FOR RADIO SHACK, A DIVISION OF TANDY CORPORATION

TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment. RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. **This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations**. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE." NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.**

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

12/84

Tandy® 1000
Mouse Controller/Calendar Board
Owner's Manual

Radio Shack®
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

Tandy 1000 Mouse Controller/Calendar Board Owner's Manual: Copyright 1985 Tandy Corporation. All Rights Reserved.

MOUSE.SYS: Copyright 1984 Tandy Corporation and Microsoft Corporation. All Rights Reserved.

PIANO.BAS: Copyright 1984 Microsoft Corporation. All Rights Reserved. Licensed to Tandy Corporation.

CLOCKGET.EXE and CLOCKSET.EXE: Copyright 1984 Tandy Corporation. All Rights Reserved.

Reproduction or use without express written permission from Tandy Corporation of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the information contained herein.

10 9 8 7 6 5 4 3 2 1

Contents

Introduction 5
About this Manual 5
Chapter 1. Installing the Hardware 7
Chapter 2. Learning about the Software 13
Chapter 3. Copying the Mouse/Calendar Software 15
Floppy Diskette Users 15
Hard Disk Users 16
Chapter 4. Initializing the Mouse Automatically 17
Creating CONFIG.SYS 17
Adding to CONFIG.SYS 17
Chapter 5. Using the System Clock 19
Initializing the Clock Automatically 19
Creating AUTOEXEC.BAT 19
Adding to AUTOEXEC.BAT 19
Setting the System Clock 20
Chapter 6. Using the Mouse 21
Mouse Anatomy 21
Mouse Surface Requirements 21
Moving the Mouse 22
Using the Mouse with PIANO 22
Chapter 7. Programming for the Mouse 25
Mouse Interface 25
Virtual Screen 26
Graphics Modes 27
Text Modes 28
Cursors 28
Buttons 29
Mouse Unit of Distance: The Mickey 29
Internal Cursor Flag 30
Making Mouse System Calls 31
From the BASIC Interpreter 31
From Assembly-Language Programs 32
From High-Level Languages 33
Sample Program 33

Chapter 8. Defining Graphics Cursors 35
Graphics Cursor Hot Spot 42
Chapter 9. Black-and-White Graphics Cursors 43
Example 43
Chapter 10. 4-Color Graphics Cursors 47
Example 47
Chapter 11. 16-Color Graphics Cursors 49
Example 49
Chapter 12. Defining Text Cursors 51
Software Text Cursor 51
Hardware Text Cursor 53
Chapter 13. Function Descriptions 55
Appendix A. PIANO Program Listing 75
Appendix B. Sample Cursors 83
Index 93

INTRODUCTION

Thank you for purchasing the Mouse Controller/Calendar Board for the Tandy® 1000. The dual-purpose Mouse/Calendar Board includes a controller for the DIGI-Mouse™ hand-held pointing device and a real-time/date clock.

Designed to be used with a variety of screen-oriented programs, the DIGI-Mouse frees you from having to use the keyboard to move the cursor and to select commands. By sliding the DIGI-Mouse across a desk top, you can guide the cursor to the words and symbols on the screen that represent the commands of a program. By pressing the buttons on the mouse, you can select the commands to be performed.

> **Note:** When using the mouse, place a piece of paper between it and the desk top. Doing this keeps the mouse clean and helps to avoid marring the desk top.

About this Manual

Chapters 1 through 6 of this manual are for anyone who wants to use the mouse. Chapter 1 discusses installation of the hardware. **Before installing the Mouse/Calendar Board and connecting the mouse, read the entire chapter carefully.** Chapters 2 through 5 discuss installation of the software and use of the clock features. It is critical that you read all of these chapters before attempting to use the mouse or clock. This will take very little time, as the chapters are short. Chapter 6 discusses use of the mouse. To get you started, it includes a sample program called PIANO.

The remainder of the manual is for programmers who want to write programs to use the mouse. Chapter 7 is an overview. Chapters 8 through 12 describe the different types of cursors and how to create them. Chapter 13 is a reference listing of the mouse function calls. Appendix A is a listing of the PIANO program. Appendix B shows how to create 8 different cursor shapes for use in your own programs.

INSTALLING THE HARDWARE

Be sure you have the following equipment:

- Tandy 1000 Mouse Controller/Calendar Board
- DIGI-Mouse (not included)
- Type CR 2320H 3-Volt Lithium Coin Cell Battery (Radio Shack Part No. 26-163, included with the board)

We highly recommend that you have the Mouse Controller/Calendar Board installed by the service technicians at your Radio Shack Service Center. Doing so not only ensures expert installation, but also enables the technicians to quickly check to be sure all the equipment is functioning properly.

If, however, you **do** decide to install the board yourself, first read through the instructions below completely, noticing all warnings about equipment handling. Then install the equipment, following the instructions exactly.

1. **Warning: Turn off and then disconnect all equipment.** If it is on, you could damage the central processing unit, as well as the controller board. After disconnecting the equipment, always wait 10 seconds before inserting or removing an option board.

2. To avoid possible static charge buildup, touch a metal object to ground yourself before you begin. Do **not** stand on a carpeted floor while doing this.

3. If the Lithium Coin Cell battery is not yet installed, slide it under the clip on the board, plus (+) side up, and snap it into place.

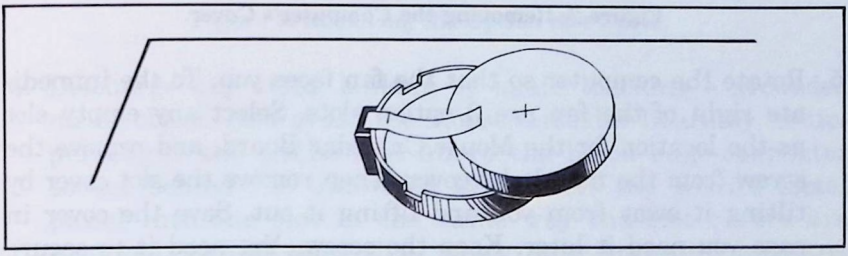


Figure 1. Installing the Battery.

If you install the battery upside-down, it does not make contact.

4. Remove the 2 screws on the front of the main unit. Then remove the cover by sliding it toward the front of the unit, parallel to the computer, as shown here:

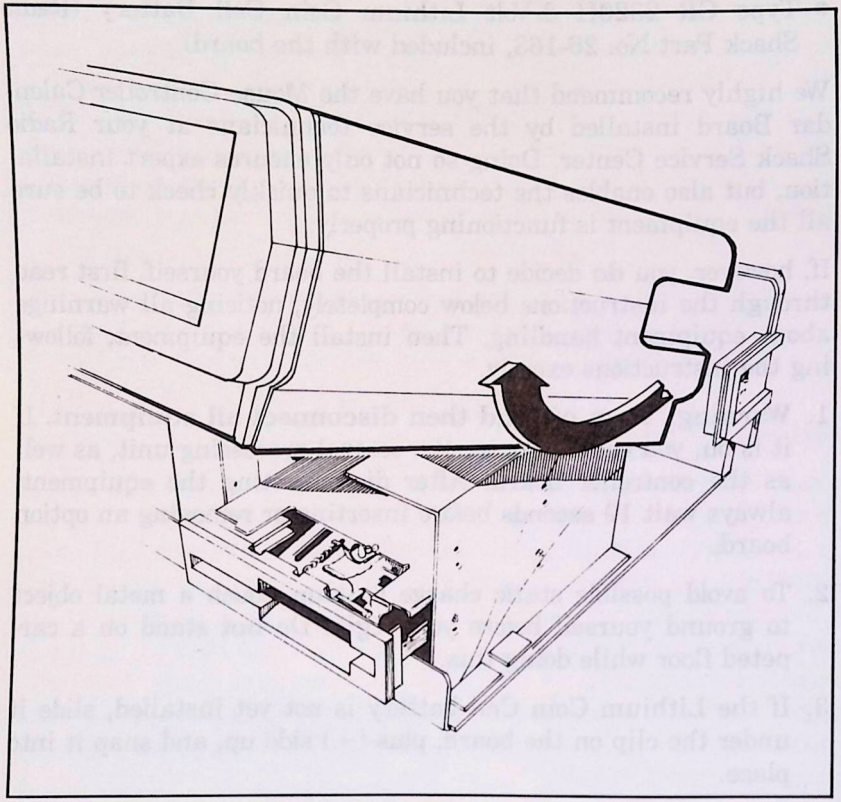


Figure 2. Removing the Computer's Cover.

5. Rotate the computer so that the fan faces you. To the immediate right of the fan are 3 option slots. Select any empty slot as the location for the Mouse/Calendar Board, and remove the screw from the metal slot cover. Then remove the slot cover by tilting it away from you and lifting it out. Save the cover in case you need it later. Keep the screw. You need it to secure the Mouse/Calendar Board.

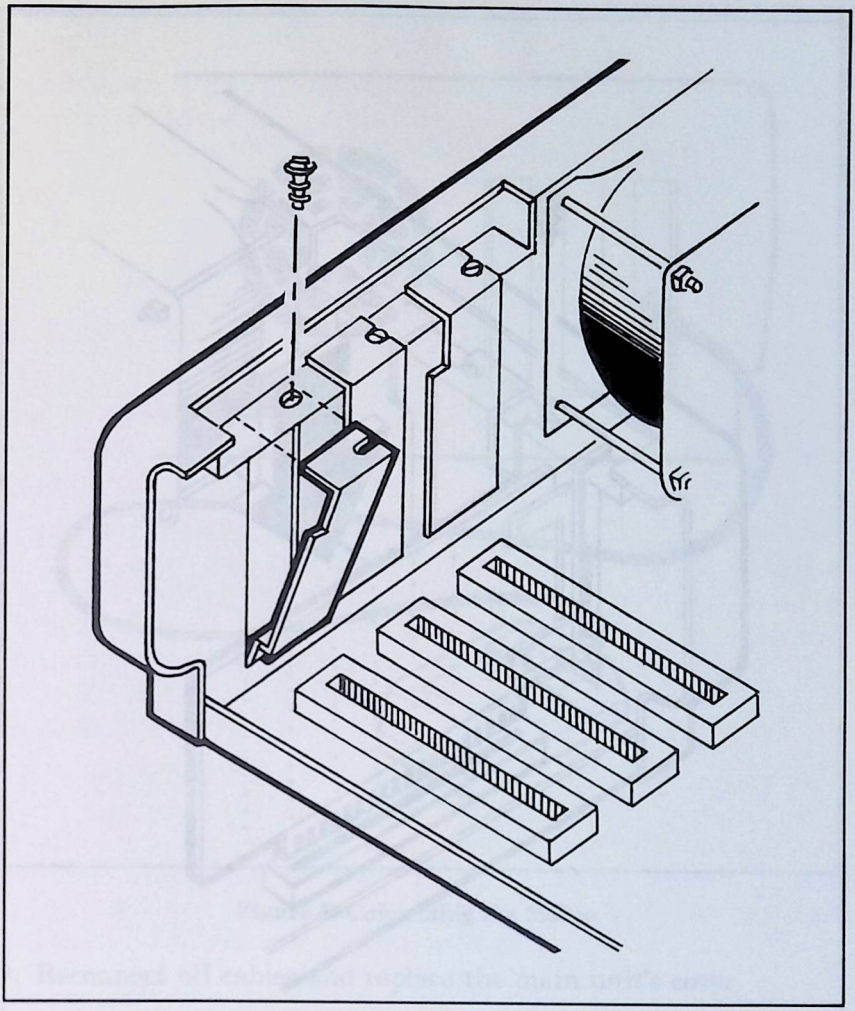


Figure 3. Removing the Option Slot Cover.

6. Discharge any static buildup by again touching a grounded metal object. Now grasp the Mouse/Calendar Board by its upper edges and position it above the black edge-connector socket behind the uncovered slot. Insert the board's metal panel into the slot in the same way the slot covers are mounted. At the same time, apply even downward pressure, engaging the edge-connector into its socket.

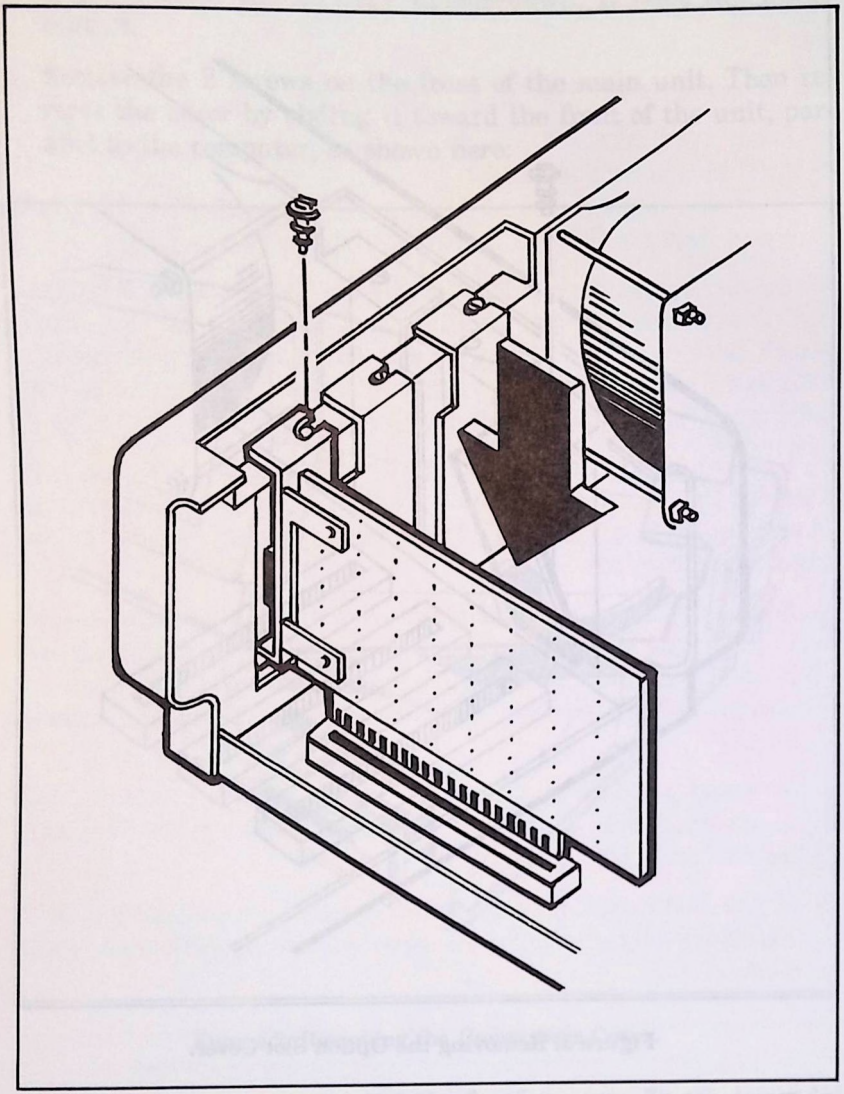


Figure 4. Inserting the Mouse/Calendar Board.

7. Replace the screw and tighten it to secure the board. Do not overtighten it.
8. Connect the cable from the DIGI-Mouse to the 9-pin connector on the Mouse/Calendar Board.

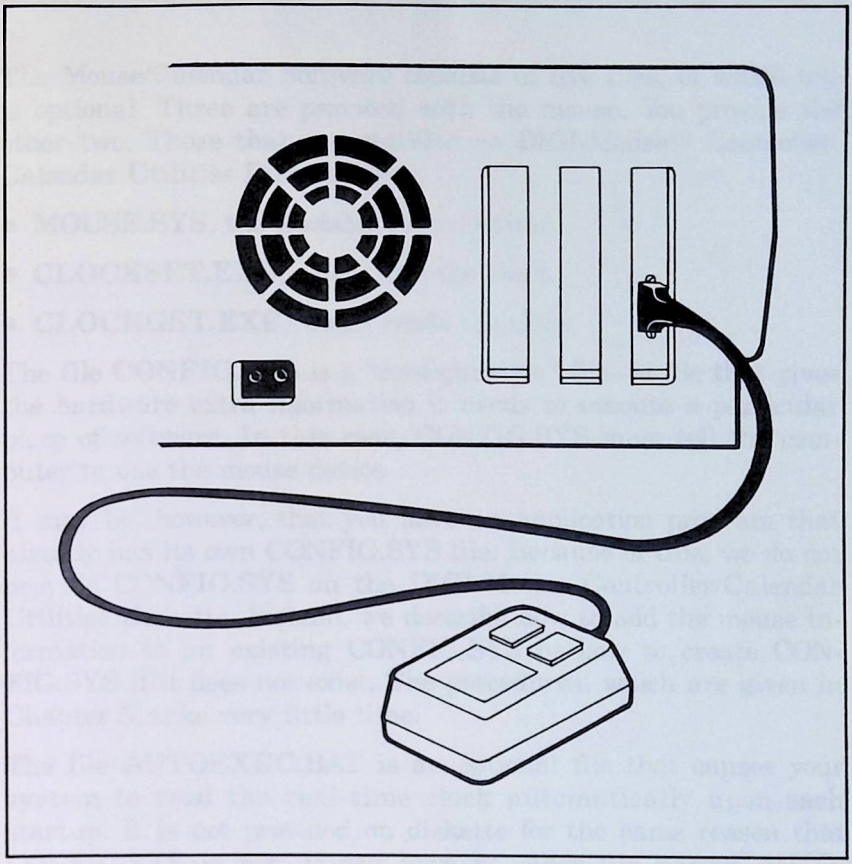
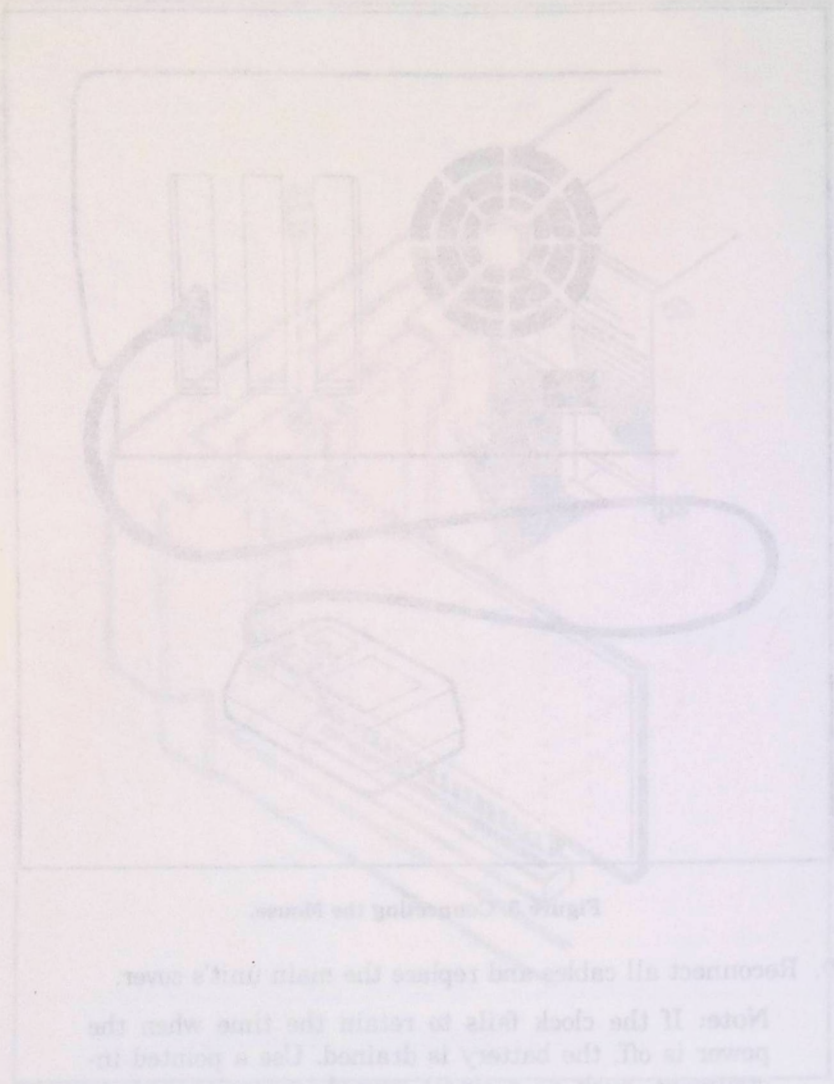


Figure 5. Connecting the Mouse.

9. Reconnect all cables and replace the main unit's cover.

   Note: If the clock fails to retain the time when the power is off, the battery is drained. Use a pointed instrument, such as a small screwdriver or an awl, to pry the battery up and out from under the clip. Replace it with another 3-volt Lithium Coin Cell Battery.



Chapter 2

LEARNING ABOUT THE SOFTWARE

The Mouse/Calendar Software consists of five files, of which one is optional. Three are provided with the mouse. You provide the other two. Those that are provided on DIGI-Mouse™ Controller/ Calendar Utilities Diskette are:

- **MOUSE.SYS**, the loadable mouse driver
- **CLOCKSET.EXE**, which sets the clock
- **CLOCKGET.EXE**, which reads the clock

The file **CONFIG.SYS** is a "configuration" file—a file that gives the hardware extra information it needs to execute a particular piece of software. In this case, CONFIG.SYS must tell the computer to use the mouse device.

It may be, however, that you have an application program that already has its own CONFIG.SYS file. Because of this, we do not provide CONFIG.SYS on the DIGI-Mouse Controller/Calendar Utilities Diskette. Instead, we describe how to add the mouse information to an existing CONFIG.SYS or how to create CONFIG.SYS if it does not exist. The procedures, which are given in Chapter 5, take very little time.

The file **AUTOEXEC.BAT** is an optional file that causes your system to read the real-time clock automatically upon each startup. It is not provided on diskette for the same reason that CONFIG.SYS is not: If you have no other file named AUTOEXEC.BAT, you can create the file; if you do have one, you can add to it. See Chapter 6 for instructions.

Chapter 3

COPYING THE MOUSE/CALENDAR SOFTWARE

The procedure for copying the mouse/calendar software varies, depending on the kind of system you have. Follow the appropriate steps below.

Floppy Diskette Users

Follow these steps once for each system diskette with which you want to use the mouse and clock. A system diskette is any diskette that contains the MS-DOS operating system. This includes any application program diskette to which you have transferred the system.

1. Turn on the Tandy 1000.
2. Insert a backup of the system diskette in Drive A. (First, be sure the diskette's write-protect notch is not covered by a tab.)
3. Insert the DIGI-Mouse Controller/Calendar Utilities Diskette in Drive B.
4. At the A> prompt, enter these commands:

```
COPY B:MOUSE.SYS A: [ENTER]
COPY B:CLOCKSET.EXE A: [ENTER]
COPY B:CLOCKGET.EXE A: [ENTER]
```

   If you later want to see the demonstration program used in Chapter 6, enter this command, also:

```
COPY B:PIANO.BAS A: [ENTER]
```

The Drive A diskette now contains all mouse/calendar files provided. Remove the DIGI-Mouse Controller/Calendar Utilities Diskette from Drive B and store it in a safe place. Then proceed to the next chapter to learn how to create CONFIG.SYS on the Drive A diskette—or how to add to it, if it already exists—so that you can use the mouse with that diskette whenever you wish.

Hard Disk Users

Follow these steps once to transfer the software to the hard disk, Drive C.

> **Note:** This procedure assumes you have formatted your hard disk and have transferred the latest version of the operating system and application programs to it. If you have not done so, see your *Hard Disk Controller Board Installation and User's Guide.*

1. Turn on your Tandy 1000 and start it up under hard disk control.
2. Insert the DIGI-Mouse Controller/Calendar Utilities Diskette in Drive A.
3. At the C> prompt, enter these commands:

```
COPY A:MOUSE.SYS C: [ENTER]
COPY A:CLOCKSET.EXE C: [ENTER]
COPY A:CLOCKGET.EXE C: [ENTER]
```

   If you later want to see the demonstration program used in Chapter 6, enter this command, also:

```
COPY A:PIANO.BAS C: [ENTER]
```

The hard disk now contains all mouse/calendar files provided. Remove the DIGI-Mouse Controller/Calendar Utilities Diskette from Drive A and store it in a safe place. Then proceed to the next chapter to learn how to create CONFIG.SYS on the hard disk—or how to add to it, if it already exists—so that you can use the mouse with all programs on your hard disk whenever you wish.

Chapter 4

INITIALIZING THE MOUSE AUTOMATICALLY

To use the mouse and the clock, the current disk must contain a CONFIG.SYS file that has in it this line:

```
DEVICE = MOUSE.SYS
```

If the current disk (Drive C for hard disk users or the diskette currently in Drive A for floppy disk users) already contains CONFIG.SYS, add to the existing file. If it does not, create the file. The procedures for doing both are described below.

Creating CONFIG.SYS

To create CONFIG.SYS on the current disk, type (at the system prompt):

```
COPY CON CONFIG.SYS [ENTER]
DEVICE = MOUSE.SYS [ENTER]
[CTRL] [Z] [ENTER]
```

To be sure the file is created and contains the proper command, type:

```
TYPE CONFIG.SYS [ENTER]
```

MS-DOS displays the command in the file.

Adding to CONFIG.SYS

To add to an existing CONFIG.SYS file, follow these steps:

1. Type:

   ```
   EDLIN CONFIG.SYS [ENTER]
   ```

2. MS-DOS displays:

   ```
   End of input file
   *
   ```

   To enter the *insert text* mode, type `I` [ENTER].

   MS-DOS displays the line number and the asterisk prompt:

   ```
   1:*
   ```

3. Type this line:

   `DEVICE = MOUSE.SYS` [ENTER]

   `DEVICE = MOUSE.SYS` is the only line required for mouse use. If, however, your application program(s) require other lines not already in the file, insert them too.

4. After typing the line(s), exit the insert mode by pressing [CTRL] [C].

5. Then type `E` [ENTER] to end the file and save all old and new lines.

To be sure the file now contains the proper command(s), type:

`TYPE CONFIG.SYS` [ENTER]

MS-DOS displays the commands in the file.

After adding to CONFIG.SYS, you must reset the system. Each time you start up from a disk that contains such a CONFIG.SYS and the file MOUSE.SYS, the mouse device driver loads automatically.

Notice that the driver requires approximately 4000 bytes of RAM. If you have an application program that requires all of RAM and does not require the mouse, delete the line `DEVICE = MOUSE.SYS` from CONFIG.SYS; then reset the system.

Chapter 5

USING THE SYSTEM CLOCK

With the Mouse/Calendar Board, you have the option of having the system read the real-time clock automatically upon each startup or reset. This information is used to set the MS-DOS system clock.

Initializing the System Clock Automatically

If you want to use this option, the current disk must contain the CLOCKGET.EXE command in a file called AUTOEXEC.BAT. If the current disk already contains an AUTOEXEC.BAT file, add the command to the existing file. If AUTOEXEC.BAT does not exist, create the file. The procedures for doing both are described below. They are similar to the procedures for creating and adding to CONFIG.SYS.

Creating AUTOEXEC.BAT

To create AUTOEXEC.BAT on the current disk, type (at the system prompt):

```
COPY CON AUTOEXEC.BAT [ENTER]
CLOCKGET.EXE [ENTER]
[CTRL] [Z] [ENTER]
```

Now that AUTOEXEC.BAT contains the CLOCKGET command, the system automatically reads (initializes) the real-time clock upon each startup or reset. Although you need to execute the command only once after each reboot, you can execute it at any time by typing `CLOCKGET` [ENTER] at the system prompt.

Adding to AUTOEXEC.BAT

To add to an existing AUTOEXEC.BAT file, follow these steps:

1. Type:

```
EDLIN AUTOEXEC.BAT [ENTER]
```

2. MS-DOS displays:

```
End of input file
*
```

Type I [ENTER] to enter the insert mode. Then type the CLOCKGET command line so that your screen looks like this:

```
1:* CLOCKGET.EXE [ENTER]
```

This is the only line required for this real-time clock function. If, however, your application program(s) require other auto commands not already in the file, insert them too.

3. When finished typing the line(s), exit the insert mode by pressing [CTRL] [C].

4. Then type E [ENTER] to end the file and save all old and new lines.

Note: For more information about batch files such as AUTOEXEC.BAT, see *Tandy 1000 MS-DOS Reference Manual.*

Setting the System Clock

You may set the real-time clock on the DIGI-Mouse/Calendar Board at any time by running CLOCKSET.

To do so, type CLOCKSET [ENTER] at the system prompt.

This causes the current system date and time to be written to the DIGI-Mouse/Calendar Board. (See the DATE and TIME commands in *Tandy 1000 MS-DOS Reference Manual.*) The clock is accurate to the nearest minute.

The clock cannot save the year; so CLOCKSET writes a file called YEAR.DAT that maintains the year. Whenever you run CLOCKGET, YEAR.DAT should be in the default (current) directory so that CLOCKGET can read it. We suggest, therefore, that you always run CLOCKGET and CLOCKSET from the root directory.

Chapter 6

USING THE MOUSE

It takes only a few minutes to learn to use the DIGI-Mouse. To help you get started, this chapter includes a demonstration run of PIANO, the sample program provided on the DIGI-Mouse Controller/Calendar Utilities Diskette.

> **Note:** The Mouse/Calendar Board uses IR3 for communicating with the CPU. You cannot use the mouse while running an application program that uses the secondary communications channel.

Mouse Anatomy

Before using the mouse, examine its working parts.

The buttons permit you to make selections when an application program presents you with a choice. When you press and release a button, the mouse passes this information to the program. A button's definition depends on the current program's definition of it.

When you slide the mouse across a hard, flat surface, the ball on the bottom of the mouse rolls in its socket. The mouse translates this rolling into directional data and passes it to the mouse software to move the cursor on the screen.

Mouse Surface Requirements

Use the mouse on any flat, hard surface, such as a desk. We recommend placing the mouse right beside the keyboard because most programs that use the mouse require a combination of mouse and keyboard input.

The mouse depends on free movement in all directions; so be sure there is adequate space for uninterrupted movement of the mouse and your arm. For most programs, a clear space of 10 by 10 inches (25.4 cm by 25.4 cm) is sufficient.

For the best performance, be sure that the surface is free of dirt, moisture, and lint. A sticky surface can prevent the ball from rolling freely. A wet surface can lead to a short in the internal circuitry, damaging the mouse. To keep the mouse as clean as possible, you may want to place a piece of paper underneath.

Notice that some accumulation of dirt and lint is unavoidable. See the *DIGI-Mouse Operation Manual* for cleaning instructions. Be sure to turn off your computer and disconnect the mouse before cleaning it.

Moving the Mouse

The mouse lets you move the cursor up, down, left, right and—unlike the keyboard—even diagonally on the screen. See the *DIGI-Mouse Operation Manual* for an illustration of how to use the mouse to control cursor movement.

Notice that the cursor moves only when the mouse moves. The location of the mouse does not matter. Thus, you can lift the mouse off the surface and return it to its starting point without returning the cursor to its starting point. This feature is useful when you are moving the cursor all the way across the screen. The move can be an accumulation of short strokes instead of one long stroke.

Using the Mouse with PIANO

If your computer is not already on, turn it on.

Floppy Diskette Users with 1-Drive Systems: Start up your system from a backup of the MS-DOS/BASIC diskette, and copy the mouse software and PIANO.BAS to that diskette. Run the program by typing `BASIC PIANO` [ENTER].

Floppy Diskette Users with 2-Drive Systems: Start up your system with a backup of the MS-DOS/BASIC diskette in Drive A and a backup of the DIGI-Mouse Controller/Calendar Utilities Diskette in Drive B. At the system prompt, switch to Drive B by typing `B:` [ENTER]. Then, load BASIC by typing `A:BASIC` [ENTER]. Run the program by typing `RUN PIANO` [ENTER].

Hard Disk Users: Be sure that Drive C contains the BASIC Interpreter—as well as the operating system, PIANO.BAS, and the mouse software—and that the system is operating under hard disk control. To load BASIC and run the program, type: `BASIC PIANO` ENTER.

Remember, if BASIC, the PIANO program, and the mouse software are not in the default drive, you must precede the filename with a drive specification.

> **Note:** To run PIANO, you must have a Tandy 1000 Monochrome Monitor or Color Monitor. If you have the monochrome monitor, change Lines 1140 and 1150, by typing:
>
> ```
> 1140 SC=1:SCREEN SC,1 ENTER
> 1150 COLOR 0,1 ENTER
> ```
>
> You can now run the program.

PIANO lets you create music at a video keyboard. The screen consists of a keyboard (21 "white" keys and 15 "black" keys) and a "quit" box in the lower right corner.

The cursor is in the middle of the screen just below the keyboard. Practice moving the cursor by moving the mouse from side to side. Notice how even a small motion of the mouse moves the cursor quickly and accurately. With only a little practice you can pinpoint even the smallest objects on the screen.

Don't be afraid to move the cursor to the edge of the screen. The screen edge forms a boundary beyond which the cursor cannot pass.

The notes of the "white" keys range from low C on the left to high B on the right. The "black" keys are the sharps and flats between these notes. To play a note, use the mouse to move the cursor over the key that you want; then press the left button. Notice that the tip of the cursor must be within the boundaries of the key.

For example, to play middle C, move the cursor to the eighth "white" key from the left and press the left button. The computer plays a middle C as long as you hold the button down and stops as soon as you release the button.

Play another note by moving the cursor to another key and pressing the left button. Notice that if you move the cursor off the piano keyboard and press the button, no note sounds.

Moving the cursor to a key and pressing the button is a method of selection. Many programs use this method to allow you to choose a program action from a menu of commands. You simply move the cursor to the word, command, or symbol that represents the action, and press the button. This method is faster and easier than typing command letters or names at the keyboard.

Now return the cursor to middle C. To play an octave higher, you can either move the cursor to the right 8 "white" keys or leave it where it is and press the mouse's right button. In PIANO, the right button always plays a note 1 octave higher than the current note.

Choosing to press one button instead of another is a method of selecting options within a given action—in this case, choosing to play the octave above instead of the note itself. Many programs use this method to permit you to select options in a command.

Starting at low C (the "white" key on the far left), press the left button and hold it down while you move the cursor across the keyboard. As the cursor moves from one key to the next, the notes change instantly and you hear a rapid series of notes. Try the right button too.

Holding a button down while moving the cursor is a method of extending an action across the screen—in this case, extending the action "play" from one key to the next. Many programs use this method to allow you to mark the range of a specific action. For example, if the action is drawing a line, you can mark the starting point, the line's path, and the ending point.

When finished playing PIANO, move the cursor to the quit box and press either button. The computer exits PIANO and displays the system prompt.

Chapter 7

PROGRAMMING FOR THE MOUSE

This chapter provides an overview of how to incorporate the DIGI-Mouse into your application programs. The first section describes the interface between the mouse software and the Tandy 1000 screen. The second section describes the steps required to make mouse system calls from BASIC, assembly, and high-level language programs.

Read all of this chapter carefully before using any of the mouse functions in your application programs.

Mouse Interface

The mouse software works with all graphics and text screen modes available with the Tandy 1000. To use the color modes, you must have a color monitor. Other than that, you can use any modes you want. (See Chapter 8, "Displaying Color and Graphics," in *Tandy 1000 BASIC Reference Manual*.)

In describing the interface between the mouse software and the screen, this section defines the following:

- Virtual screen
- Types of cursors that you can create for use with the mouse—including graphics, software text, and hardware text cursors
- Mouse buttons
- Mouse unit of distance
- Internal cursor flag

For your convenience, details on how to create the cursors are broken up into separate chapters. Chapter 8 includes the general procedure for creating a graphics cursor. Chapters 9, 10, and 11 give specific examples for creating black-and-white, 4-color, and 16-color graphics cursors. Chapter 12 explains how to create a software text cursor or a hardware text cursor. Chapters 8 and 12 are based on many of the same concepts; therefore, you should read both chapters, regardless of the type of cursor you wish to create.

Virtual Screen

For simplicity, the mouse software operates on the Tandy 1000 screen as if it were a *virtual screen* of 128,000 points arranged in a matrix of 640 horizontal by 200 vertical points, as shown here:

```
0,0 ..........................................639,0
0,1 ..........................................639,1
0,2 ..........................................639,2
 .                                              .
 .                                              .
 .                                              .
 .                                              .
0,199 ..................................... 639,199
```

Figure 6. Coordinate System of Virtual Screen

A pixel in the upper left corner of the screen has the virtual screen coordinates (0,0) and a pixel at the center of the screen has the virtual screen coordinates (320,100). (The horizontal coordinate is given first.)

Do not confuse the virtual screen with the screen sizes of the various screen modes. **Changing the screen mode does not change the virtual coordinates. It merely sets a limit on which of them you can specify.** The limits and the causes of them are explained in detail in the "Graphics Modes" and "Text Modes" subsections.

Because the mouse software uses virtual coordinates when referring to an object on the screen, any application program that you write for the mouse must also use virtual coordinates. When making a mouse function call, be sure all coordinates you specify are legal for the given screen mode. (Legal coordinates are defined in "Graphics Modes" and "Text Modes" below.) The coordinates returned by the mouse software are always legal for the given mode.

The following table summarizes the screen mode attributes. Before proceeding, study it carefully, taking particular note of the number of bits per pixel for the graphics modes.

BASIC Mode	BIOS Mode	Screen Size	Color	Cursor Size (pxls x scan lines)	Bits per Pixel (bits/pxl)	Actual Cursor Size (bits)	Bytes Per Line	Graphics Memory
0	0	40x25	b/w	N/A (alpha)	—	—	—	2k
0	1	40x25	color	N/A (alpha)	—	—	—	2k
0	2	80x25	b/w	N/A (alpha)	—	—	—	2k
0	3	80x25	color	N/A (alpha)	—	—	—	2k
1,4	4	320x200	4	8x16	2	256	80	16k
1,4	5	320x200	4(b/w)	8x16	2	256	80	16k
2	6	640x200	2(b/w)	16x16	1	256	80	16k
Not valid	7							
3	8	160x200	16	8x8	4	256	80	16k
5	9	320x200	16	8x8	4	256	160	32k
6	10	640x200	4	8x16	2	256	160	32k

Table 1. Screen Modes.

Note: The number of available colors determines the number of bits per pixel for a graphics mode. For example, a 16-color mode requires 4 bits to define each pixel.

Graphics Modes

Depending on the graphics mode you choose, you might be able to use all pairs of virtual coordinates or you might be restricted to every other or every fourth pair. This is because the number of bits per pixel varies with the screen mode, as you can see from studying Table 1.

In BASIC Screen Modes 2 and 6, each pixel on the actual screen corresponds to 1 point on the virtual screen. Therefore, the full range of coordinates from (0,0) to (639,199) is permitted.

In BASIC Screen Modes 1, 4, and 5, the number of pixels on the screen is ½ that on the virtual screen (320 × 200 = 64000). Thus, each pixel on the screen corresponds to 2 points on the virtual screen. To compensate, the mouse software uses even-numbered horizontal coordinates only.

In BASIC Screen Mode 3, the number of pixels is 1/4 that on the virtual screen (160 × 200 = 32000). Thus, each pixel on the screen corresponds to 4 points on the virtual screen. To compensate, the mouse software uses every other even-numbered horizontal coordinate only.

Text Modes

In 80-column text mode (BASIC Screen 0), only characters are permitted on the screen. There are still 128,000 pixels on the screen, each corresponding one-to-one with the points on the virtual screen, but the individual pixels in a character cannot be accessed. Because of this, the mouse software uses only 1 pair of coordinates, that of the pixel in upper left corner of the character, to refer to a character.

Since each character in this mode is an 8- by 8-pixel group, both the horizontal coordinate and the vertical coordinate are multiples of 8. For example, the character in the upper left corner of the screen has the coordinates (0,0), the next character to the right has the coordinates (8,0), and so on.

In 40-column text mode (BASIC Screen 0), the mouse software again uses the coordinates of only 1 pixel in a character to refer to the location. But the number of pixels is ½ that in 80-column text mode. To compensate, the mouse software uses horizontal coordinates that are multiples of 16. For example, the character in the upper left corner of the screen still has the coordinates (0,0), but the character next to it has the coordinates (16,0). The vertical coordinates are still multiples of 8.

Cursors

The mouse has 3 cursors—a graphics cursor, a software text cursor, and a hardware text cursor. The graphics cursor is a shape (for example, an arrow) that moves over the images on the screen. The software text cursor is an alphanumeric character or other character attribute, such as an underscore, that moves from character to character on the screen. The hardware text cursor is a flashing block, half-block, or underscore that moves from character to character.

Although only 1 cursor can be on the screen at a time, you can switch back and forth among the various cursors.

Chapter 8 discusses the graphics cursor in detail and includes the general procedure for creating a graphics cursor. Chapters 9, 10, and 11 include procedures for creating sample graphics cursors in black and white, 4 colors, and 16 colors.

Chapter 12 discusses the text cursors.

Buttons

The mouse functions read the status of the buttons on the mouse and keep a count of the number of times the buttons are pressed and released.

The button's status is *pressed* if the button is down and *released* if the button is up. When a function returns the status of the buttons, it returns an integer value in which the first 2 bits are set or cleared. Bit 0 represents the status of the left button, and Bit 1 represents the status of the right button. If a bit is set (equal to 1), the button is down. If a bit is clear (equal to 0), the button is up.

The mouse software has internal counters to keep track of the number of times a button is pressed and released. The software increments a counter each time the corresponding button is pressed or released. The software sets a counter to zero after a reset (Function 0) or after a counter's contents are read (Functions 5 and 6).

Mouse Unit of Distance: The Mickey

The mouse hardware translates the motion of the ball in the mouse into values that express the direction and duration of the motion. The values are given in a unit of distance called a *mickey*, which is approximately 1/80 of an inch (0.32 mm).

When you slide the mouse across a desk top, the mouse hardware passes the software a horizontal and a vertical *mickey count* (the number of mickeys the mouse ball has rolled horizontally and vertically). The software uses the mickey count to move the cursor a certain number of pixels on the screen.

The number of pixels moved does not have to correspond one-to-one with the number of mickeys the ball rolled. The mouse software defines a *sensitivity* for the mouse, which is a ratio of the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines the rate at which the cursor moves on the screen.

You can define the sensitivity of the mouse by passing a mickey count to Function 15 of the mouse system calls. The count can be any value from 1 to 32767. For example, if you send a count of 8, the sensitivity is 8 mickeys per 8 pixels. That is, the cursor moves 1 pixel for each mickey the ball rolls, or 1 character for every 8 mickeys the ball rolls.

Internal Cursor Flag

The mouse software maintains an internal flag that determines whether the cursor is visible or not. When the flag's value is 0, the cursor is displayed. When the flag's value is any other number, the cursor is hidden. Initially, the flag's value is −1.

The flag is not directly accessible to your program. To change the flag's value, you must use FlagInc and FlagDec (Functions 1 and 2) in the mouse system calls. FlagInc increments the flag's value by 1; FlagDec decrements it by 1.

You can call FlagInc and FlagDec any number of times. Remember, however, that each call to one function requires a subsequent call to the other to restore the flag's previous value. For example, suppose the cursor is on the screen and you make 1 call to FlagDec to remove it and then you make 4 more FlagDec calls. You must make 5 calls to FlagInc to get the cursor back on the screen.

Using GetMouseStat/Reset (Function 0) resets the flag to −1, as does changing the screen mode.

Making Mouse System Calls

This section describes how to make mouse system calls from the BASIC Interpreter, from assembly-language programs, and from programs in a high-level language compilers such as COBOL, FORTRAN, Pascal, and BASIC. The statements and/or instructions required to make the calls depend on the language of your application program.

You can also let the mouse software call a subroutine in your program whenever a specific condition occurs. When this capability is enabled, the mouse software interrupts whatever process is going on and passes execution control to the subroutine that you have specified in Function 12 of the mouse system calls. For details, see the description of Function 12.

From the BASIC Interpreter

To make a mouse system call from a BASIC program running under the BASIC Interpreter, you must:

1. Assign the offset and segment address of the mouse software to a pair of integer variables in your program. The mouse entry offset and segment address are in memory. To get these values, insert the following statements into your program:

```
10  DEF SEG=0
20  MSEG=256*PEEK(51*4+3)+PEEK(51*4+2)
30  MOUSE=256*PEEK(51*4+1)+PEEK(51*4)+2
40  IF MSEG AND MOUSE THEN 60
50  PRINT "Mouse Driver not found": END
60  DEF SEG=MSEG
```

   Be sure that the statements appear before any calls to mouse functions.

2. Use the CALL statement to make the call. The statement should have the form:

```
CALL MOUSE(M1%,M2%,M3%,M4%)
```

MOUSE is the variable containing the entry offset of the mouse software, and M1%, M2%, M3%, and M4% are the names of the integer variables you have chosen for the parameters in this call. All 4 parameters must appear in the CALL statement even if no value is assigned to 1 or more of them. These must be integer variables. Constants and noninteger variables are not allowed.

To ensure that the variables are integer variables, use the per cent sign (%) as the variable name. You may also use the DEFINT statement at the beginning of your program. For example, this statement defines all variables as integers:

```
10 DEFINT A-Z
```

With this statement at the beginning of the program, the per cent sign is optional.

Example:

Assuming that the variable MOUSE has the mouse software offset, use the following statements to set the cursor position to 320 (horizontal) and 100 (vertical):

```
100  '
200  ' Set cursor position to (320,100)
300  '
400  M1% = 4    ' function number is 4
500  M3% = 320  ' horizontal coordinate
600  M4% = 100  ' vertical coordinate
700  CALL MOUSE(M1%,M2%,M3%,M4%)
```

From Assembly-Language Programs

To make mouse system calls from an assembly-language program, you must:

1. Load the AX, BX, CX, and DX registers with the parameter values.
2. Execute software interrupt 51 (33H).

The AX, BX, CX, and DX registers correspond to the M1%, M2%, M3% and M4% parameters defined for the BASIC program. Values returned by the mouse functions are placed in the registers.

Example:

Use the following instructions to set the cursor position to 320 (horizontal) and 100 (vertical):

```
*
* Set Cursor to Location (320,100)
*
MOV AX,4        ;function #4
MOV CX,320      ;set horizontal to 320
MOV DX,100      ;set vertical to 100
INT 51          ;interrupt to mouse
```

This call has the same effect as the call from the BASIC program shown in the previous example.

> **Note:** When making a mouse system call in assembly language, Functions 9 and 12 expect a somewhat different value for the fourth parameter than when calling from a BASIC program. See the description of these functions for details.

From High-Level Languages

You can make calls from compiled COBOL, FORTRAN, Pascal, and BASIC language programs. To do so, follow these steps:

1. Write an assembly-language subroutine to call the mouse driver (as explained in the preceding section).
2. Call the routine from the desired high-level language. (For more information, refer to the manual provided with the high-level language.) The assembly-language routine passes arguments between the high-level language routine and the mouse driver.

Sample Program

To help you learn how to use the mouse system calls, Appendix A contains the listing of the PIANO demonstration program described in Chapter 6. We recommend that you read the listing and refer to Chapter 13 for details on the operation of each function. The PIANO program listing is also in the file PIANO.BAS on the DIGI-Mouse Controller/Calendar Utilities Diskette. To use the program, load and run it as described in Chapter 6.

DEFINING GRAPHICS CURSORS

The graphics cursor is the cursor used when the computer is in graphics mode. It is a block of 256 bits. The shape of the block is the product of the cursor size and the number of bits per pixel. Thus, for BASIC Screen Modes 3 and 5, it is a 32- by 8-bit block. For all other graphics modes, it is a 16- by 16-bit block.

Here is the general procedure for creating a graphics cursor.

1. Select the BASIC screen mode, based on the resolution and the number of colors you want. Notice the number of bits per pixel for that screen mode.

   The number of available colors determines the number of bits per pixel. A 16-color set requires that the screen have 4 bits per pixel. A 4-color set requires 2 bits per pixel. A 2-color set (black and white) requires only 1 bit per pixel.

2. Select the colors that you want to use; then, determine the binary equivalents of their color numbers.

   In black-and-white mode, black has the binary value of 0 and white has the binary value of 1.

   In 4-color mode, Colors 0 through 3 are available, as follows:

Color Number	Binary Value	Default Color
0	00	Black
1	01	Cyan
2	10	Magenta
3	11	White

Table 2. Binary Values for 4-Color Mode.

In 16-color mode, Colors 0 through 15 are available, as follows:

Color Number	Binary Value	Default Color
0	0000	Black
1	0001	Blue
2	0010	Green
3	0011	Cyan
4	0100	Red
5	0101	Magenta
6	0110	Brown
7	0111	Gray
8	1000	Dark gray
9	1001	Light blue
10	1010	Light green
11	1011	Light cyan
12	1100	Light red
13	1101	Light magenta
14	1110	Yellow
15	1111	White

Table 3. Binary Values for 16-Color Mode.

In the 4- and 16-color modes, you can use the PALETTE and PALETTE USING statements to change the colors in the palette. (See *Tandy 1000 BASIC Reference Manual* for more information.)

3. If you are using Screen Mode 3 or 5, sketch a 32 by 8 grid. If you are using any other graphics mode, sketch a 16 by 16 grid. On this grid, fill in the binary values needed to produce the colors you want.

   For example, to produce a solid white cursor (binary 1) with a narrow black (binary Ø) border in Screen Mode 2, your cursor block must contain the following values:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7. Sample Cursor Block for B/W Mode.

Note: The sample cursor blocks are shaded to help you visualize the cursor. You may want to do this yourself when sketching cursor blocks.

To produce a cursor that has white (binary 11) and magenta (binary 10) vertical stripes and a black (binary 00) border in Screen Mode 1, your cursor block must contain the following values, assuming the palette contains the default colors:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	1	1	1	0	1	1	1	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8. Sample Cursor Block for 4-Color Mode.

4. Calculate the *screen mask* and the *cursor mask* values needed to arrive at the cursor you sketched.

   The screen mask and the cursor mask are 2 arrays that the mouse software operates on to produce a cursor. They are either 32 bits by 8 bits or 16 bits by 16 bits, depending on the screen mode. Each bit in each mask corresponds to a bit in the cursor block.

   Here is how the software operates on the masks. First, the software logically ANDs the screen mask with the original 256 bits that make up the screen. Then, it logically XORs the cursor mask with the result of the AND operation.

   Tables 4 and 5 summarize the results of all possible AND and XOR operations. Table 6 is an example of how the operations affect individual screen bits.

Operand 1	Operand 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

Table 4. AND Operations.

Operand 1	Operand 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Table 5. XOR Operations.

Screen Bit	Screen Mask Bit (AND)	Result of AND	Cursor Mask Bit (XOR)	Result of XOR
0	0	0	0	0
1	0	0	0	0
0	0	0	1	1
1	0	0	1	1
0	1	0	0	0
1	1	1	0	1
0	1	0	1	1
1	1	1	1	0

Table 6. Mask Bit Values and Screen Results.

A close look at Table 6 reveals that you can essentially ignore the value of the original screen bit. The screen masks and cursor masks always interact as follows, regardless of the value of original screen bit:

If the screen mask bit (AND) is:	And the cursor mask bit (XOR) is:	The result after the AND and XOR operations is:
0	0	0
0	1	1
1	0	the same as the original screen bit value (1 if the original value is 1; 0 if it is 0)
1	1	the opposite of the original screen bit value (1 if the original value is 0; 0 if the original value is 1)

Table 7. Mask Bit Value and Screen Result (Condensed).

Now that you know the results of the particular combinations of screen mask bit values and cursor mask bit values, you can easily work backward from your sketched cursor to determine the mask bit values needed to arrive at it.

Consider again the black, white, and magenta cursor described in Step 3. Each black pixel has the binary value 00. Therefore, each corresponding screen mask pixel (2 bits) must contain the value 00, and each corresponding cursor mask pixel must contain the value 00. The screen mask and cursor mask values needed to produce a white pixel are 00 and 11, respectively. Those needed to produce a magenta pixel are 00 and 10. Thus, the masks for this example are as shown:

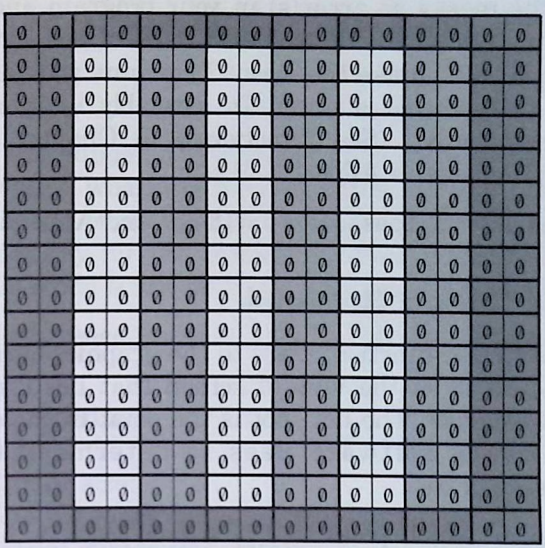


Figure 9. Screen Mask for Cursor in 4-Color Mode.

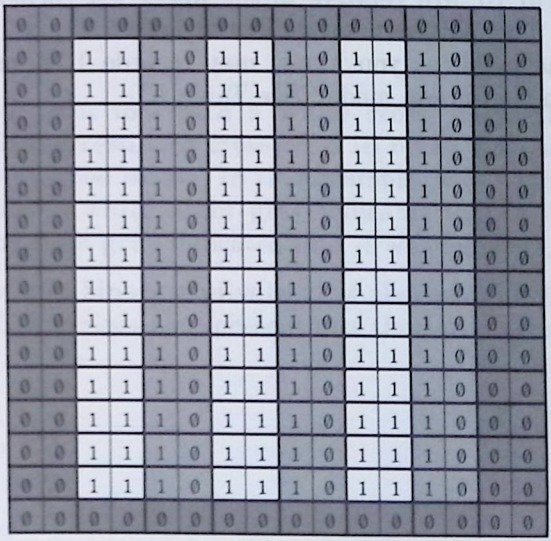


Figure 10. Cursor Mask for Cursor in 4-Color Mode.

5. Define the masks as array(s) in your program and pass them as parameters in a call to Function 9 of the mouse system calls. In BASIC, make the array 2 columns wide and use each column to represent a mask. In assembly language, make 2 contiguous arrays. (See Function 9 in Chapter 13 and the sample programs in Appendix B.)

For sample cursor shapes, see Function 9 and Appendix B.

Graphics Cursor Hot Spot

Whenever a mouse function refers to the graphics cursor location, it gives the point on the virtual screen that lies directly under the cursor's *hot spot*. The hot spot is the point in the cursor block that the mouse software uses to determine the cursor coordinates.

You can define which point in the cursor block will be the hot spot by passing the horizontal and vertical coordinates of the point to Function 9. The coordinates, which must be within the range -16 to 16, are relative to the upper left corner of the cursor block. The hot spot value should define 1 pixel within the cursor. The pixel, as you know, may include 1, 2, or 4 bits, depending on the screen mode. (See Table 1.)

Chapter 9

BLACK-AND-WHITE GRAPHICS CURSORS

Chapter 8 introduced the general procedure for determining mask bit values when creating graphics cursors. This chapter gives an example of how to apply that procedure when creating a cursor specifically for the black-and-white mode, BASIC Screen Mode 2. This mode is the only one that has 1 bit per pixel.

For ease of reference, here is a list of the attributes for Screen Mode 2:

Screen size:	640 x 200
Color set:	2 (b/w)
Cursor size:	16 x 16
Bits per pixel:	1
Actual cursor size (bits):	256
Bytes per line:	80
Graphics memory:	16k

Example

Suppose you want to create a white cross, outlined in black. First, sketch the resulting cross on a 16 by 16 grid, shading the different areas. (See below.) In this example, white indicates that the area will be white, light gray indicates that it will be unchanged, and dark gray indicates that it will be black.

Then, fill in the binary equivalents of the colors (0 for black, 1 for white). Use periods to represent bits that are unchanged.

The grid now looks like this:

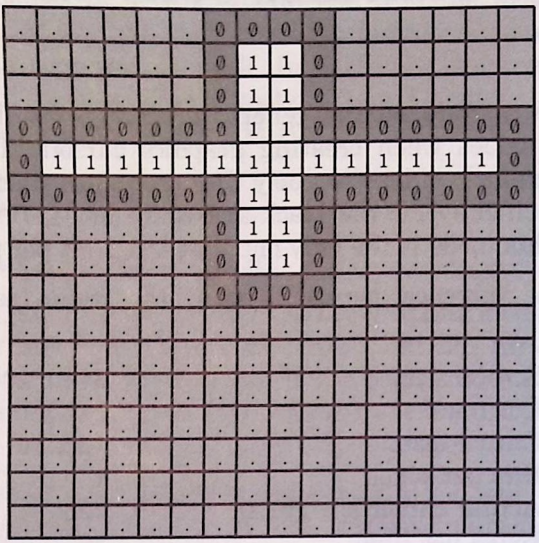


Figure 11. Cursor Block for B/W Cross.

Screen Mask Bit (AND)	Cursor Mask Bit (XOR)	Result after AND and XOR
0	0	0 (black)
0	1	1 (white)
1	0	the same as the original screen bit value (1 if the original value is 1; 0 if it is 0)
1	1	the opposite of the original screen bit value (1 if the original value is 0; 0 if the original value is 1)

Table 8. Mask Bit Values for B/W Modes.

Referring to Table 8, determine the mask bit values needed to achieve the resulting values. They are as follows:

1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Screen Mask

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Cursor Mask

Figure 12. Mask Bit Values for B/W Cross.

Now you can define the masks as arrays in your program and pass them as parameters in a call to Function 9.

> **Note:** The sample cursor in this chapter includes areas that are black, white, or unchanged from the original screen bits. For an example of a cursor area that is the inverse of the original, see the cursor definitions in the PIANO program in Appendix A.

4-COLOR GRAPHICS CURSORS

Chapter 8 introduced the general procedure for determining mask bit values when creating graphics cursors. This chapter gives an example of how to apply that procedure when creating a cursor specifically for a 4-color mode (BASIC Mode 1, 4, or 6). These modes all have 2 bits per pixel.

Screen Modes 1 and 4 have the following attributes:

Screen size:	320 x 200
Color set:	4
Cursor size:	8 x 16
Bits per pixel:	2
Actual cursor size (bits):	256
Bytes per line:	80
Graphics memory:	16k

Screen Mode 6 has these attributes:

Screen size:	640 x 200
Color set:	4
Cursor size:	8 x 16
Bits per pixel:	2
Actual cursor size (bits):	256
Bytes per line:	160
Graphics memory:	32k

Example

Suppose you want to create a cursor the left half of which is white and the right half of which is magenta. First, sketch the resulting cursor on a 16 by 16 grid, shading one half. (See below.) In this example, white indicates the area that will be white and gray indicates the area that will be magenta.

Then, referring to Table 9, fill in the binary equivalents of the colors.

Screen Mask Bits	Cursor Mask Bits	Result after AND and XOR	Color Number	Default Color
00	00	00	0	Black
00	01	01	1	Cyan
00	10	10	2	Magenta
00	11	11	3	White

Table 9. Mask Bit Values for 4-Color Modes.

The grid now looks like this:

1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0

Figure 13. Cursor Block for Cursor in 4-Color Modes.

Determine the values for the screen mask and the cursor mask. This is easy. If you use 0's in the screen mask, the cursor mask values are the same as the resulting values.

Now you can define the masks as arrays in your program and pass them as parameters in a call to Function 9.

16-COLOR GRAPHICS CURSORS

Chapter 8 introduced the general procedure for determining mask bit values when creating graphics cursors. This chapter gives an example of how to apply that procedure when creating a cursor specifically for a 16-color mode (BASIC Mode 3 or 5). These modes all have 4 bits per pixel.

Screen Mode 3 has these attributes:

Screen size:	160 x 200
Color set:	16
Cursor size:	8 x 8
Bits per pixel:	4
Actual cursor size (bits):	256
Bytes per line:	80
Graphics memory:	16k

Screen Mode 5 has these:

Screen size:	320 x 200
Color set:	4
Cursor size:	8 x 8
Bits per pixel:	4
Actual cursor size (bits):	256
Bytes per line:	160
Graphics memory:	32k

Example

Suppose you want to create a cursor that has vertical stripes in a spectrum of colors (red, light red, cyan, light green, green, light blue, blue, and magenta).

First, sketch the resulting cursor on a 32 by 8 grid, filling in the binary equivalents of the colors. Refer to Table 10 for the binary equivalents.

Screen Mask Bits (AND)	Cursor Mask Bits (XOR)	Result after AND and XOR	Color Number	Default Color
0000	0000	0000	0	Black
0000	0001	0001	1	Blue
0000	0010	0010	2	Green
0000	0011	0011	3	Cyan
0000	0100	0100	4	Red
0000	0101	0101	5	Magenta
0000	0110	0110	6	Brown
0000	0111	0111	7	Gray
0000	1000	1000	8	Dark gray
0000	1001	1001	9	Light blue
0000	1010	1010	10	Light green
0000	1011	1011	11	Light cyan
0000	1100	1100	12	Light red
0000	1101	1101	13	Light magenta
0000	1110	1110	14	Yellow
0000	1111	1111	15	White

Table 10. Mask Bit Values for 16-Color Modes.

The grid should look like this:

0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	1

Figure 14. Cursor Block for Cursor in 16-Color Mode.

Determine the values for the screen mask and the cursor mask. This is easy. If you use 0's in the screen mask, the cursor mask values are the same as the resulting values.

Now you can define the masks as arrays in your program and pass them as parameters in a call to Function 9.

DEFINING TEXT CURSORS

In addition to the graphics cursor, you can create 2 kinds of text cursors for use with the mouse. This chapter describes how to do this. To understand it well, you should also read Chapter 9, which contains a detailed explanation of screen masks and cursor masks.

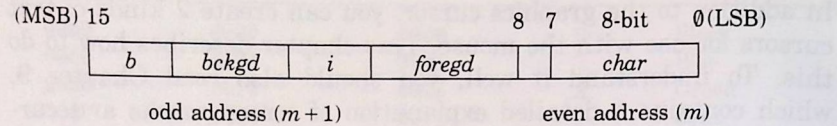
Software Text Cursor

The software text cursor is used when the computer is in 40- or 80-column text mode. The text cursor affects the appearance of the characters on the screen. Unlike the graphics cursor, the text cursor usually does not have a shape of its own. Instead, it changes character attributes such as the foreground and background color, intensity, and underscoring of the character directly under it. If the cursor does not have a shape of its own, it is one of the 256 ASCII characters listed in *Tandy 1000 BASIC Reference Guide*.

The effect of the text cursor on the character under it is defined by two 16-bit values called the screen mask and the cursor mask. The screen mask determines which attributes of the character on the screen are to be preserved. The cursor mask determines how these attributes are to be altered to yield the cursor.

To create the cursor, the mouse software operates on the data that defines each character on the screen. The software first logically ANDs the screen mask and the 16 bits of screen data for the character under the cursor. It then logically XORs the cursor mask and the result of the AND operation.

In both the 40-column and 80-column text modes, the 16 bits of screen data for each character take the following form:



b	sets blinking (1) or nonblinking (0) character
bckgd	sets the background color
i	sets high (1) or medium (0) intensity
foregd	sets the foreground color
char	specifies the ASCII value of the character, in binary form

Figure 15. Format for Software Text Cursor.

The screen and cursor masks are divided into the same fields as shown above; so, the value of these fields in the screen and cursor masks defines the new attributes of the character when the cursor is over it.

For example, to invert the foreground and background colors but keep the cursor character the same, the results of the AND and XOR operations must be as given in the table below. Therefore, the mask bit values must be as given. (See Table 7 in Chapter 8 for information on the interaction of screen mask and cursor mask bits.)

	b	*bckgd*	*i*	*foregd*	*char*	=
screen mask	0	111	0	111	11111111	&H77FF
cursor mask	0	111	0	111	00000000	&H7700
result	0	opposite	0	opposite	same	

Table 11. Mask Bit Values for Inverted Software Text Cursor.

Note: For your convenience, the table includes the hexadecimal forms of the screen mask and cursor mask values needed to produce the character.

To specify a blinking, high-intensity 0 as the software text cursor, the results and mask bit values must be these:

	b	*bckgd*	*i*	*foregd*	*char*	=
screen mask	0	111	0	111	00000000	&H7700
cursor mask	1	000	1	000	00110000	&H8830
result	1	same	1	same	00110000	

Table 12. Mask Bit Values for Blinking 0 Software Text Cursor.

You can define the value of the screen mask and the cursor mask by passing their values as parameters in Function 10 of the mouse system calls. For an example, see the description of Function 10 in Chapter 13.

Whenever a mouse function refers to the text cursor location, it gives the coordinates of the character under the cursor. The text cursor does not have a hot spot.

Hardware Text Cursor

The hardware text cursor is another cursor you can use when the computer is in text mode. It is actually the computer's own cursor, the one that follows the system prompt on the screen. The mouse software lets you adapt this cursor for your own use.

The hardware cursor is 8 pixels wide and from 1 to 8 pixels tall. Each horizontal set of pixels forms a line, called a *scan line.* There are from 1 to 8 scan lines. A scan line can be on or off. If a line is on, it appears as a flashing bar on the screen. If it is off, it has no effect on the screen. You may turn on any number of adjacent scan lines. Gaps between scan lines are not allowed. This gives the hardware text cursor a characteristic box or underscore shape.

You can define which lines are on and which are off by passing the number of the first and last lines in the cursor to Function 10 of the mouse system calls.

FUNCTION DESCRIPTIONS

This chapter describes the input, output, and operation of the 16 Tandy 1000 mouse functions. The following is a list of the functions:

Number	Function
0	Get Mouse Installation Status/Reset
1	Increment Cursor Flag
2	Decrement Cursor Flag
3	Get Mouse Position and Button Status
4	Set Mouse Cursor Position
5	Get Button Press Information
6	Get Button Release Information
7	Set Minimum and Maximum Horizontal Position
8	Set Minimum and Maximum Vertical Position
9	Set Graphics Cursor Block
10	Set Text Cursor
11	Read Mouse Motion Counters
12	Set User-Defined Subroutine Input Mask
13	Reserved
14	Reserved
15	Set Mickey/Pixel Ratio

Each description specifies the parameters required to make the call (entry conditions) and the expected return values (exit conditions), any special considerations to be taken, and an example. All examples show BASIC program segments.

In the function descriptions, the parameter names M1%, M2%, M3%, and M4% are dummy variable names. When making a call, use the names of the variables that you want to pass.

The dummy variable names include the per cent sign (%) to emphasize that only integer variables can be used as parameters. Constants, single-precision variables, and double-precision variables are not allowed.

If the function description does not specify an entry condition for a parameter, you need not supply a value before making the call. If the function description does not specify an exit condition value for a parameter, the parameter's value after the call is the same as before the call.

> **Caution:** The mouse software does not check entry condition values; so, be sure that the values you assign to the parameters before making a call are correct for the given screen mode. If you assign incorrect values, you receive unpredictable results.

GetMouseStat/Reset — Function Call 0

Get Mouse Installation Status/Reset

Returns the current installation status of the mouse hardware and software and resets the mouse driver to the following default parameters:

Function	Parameter
Cursor position	Screen center
Internal cursor flag	-1
Graphics cursor shape/hot spot	Arrow/(-1,-1)
User-defined call mask	All zeroes
Horizontal mickey to pixel ratio	8 to 8
Vertical mickey to pixel ratio	16 to 8
Horizontal min./max. cursor position	0/639
Vertical min./max. cursor position	0/199

Entry Conditions

M1% = 0

Exit Conditions

M1% = *mouse status*
M1% = 0: not installed
M1% = -1: installed
M2% = *number of buttons* (always 2)

Example

To find out if the mouse hardware and software are installed, and to reset the mouse driver to its default values, you can use these lines at the beginning of your program:

```
000  '
100  '  Is Mouse present?  If not, error.
200  '
300  M1% = 0
400  CALL MOUSE(M1%,M2%,M3%,M4%)
500  IF NOT(M1%)THEN PRINT "Mouse not
installed.":END
```

FlagInc — Function Call 1

Increment Cursor Flag/Show Cursor

Increments the internal cursor flag by 1. The maximum value you can increment to is 0. Whenever the flag value is 0, the cursor is visible. Whenever it is any negative value, the cursor is invisible.

The current value of the internal cursor flag depends on the number of calls you have made to FlagInc and FlagDec (Function 2). (See "Internal Cursor Flag" in Chapter 7.) Whenever you make a call to GetMouseStat/Reset (Function 0), the flag returns to its default value of -1. Thus, one way to cause the cursor to appear is to make a call to GetMouseStat/Reset and then immediately make a call to FlagInc. The other is to make the same number of calls to FlagInc as you have made to FlagDec since the cursor was last visible.

Entry Conditions

M1% = 1

Exit Conditions

None

Example

If the flag value is -1, you can use these statements to display the cursor:

```
100  '
200  '  Show the cursor.
300  '
400  M1% = 1
500  CALL MOUSE(M1%,M2%,M3%,M4%)
```

FlagDec — Function Call 2

Decrement Flag/Hide Cursor

Decrements the internal cursor flag by 1. Whenever the cursor flag value is negative, the cursor is invisible, although it still tracks the movement of the mouse. Whenever the flag value is 0, the cursor is visible. Thus, if the flag is 0, a call to Function 2 removes the cursor from the screen.

Remember that each call to FlagDec requires a subsequent call to FlagInc (Function 1) to restore the flag to its previous value. (See "Internal Cursor Flag" in Chapter 7.)

Use this function before modifying any position of the screen containing the cursor. This prevents the cursor from possibly affecting the data written to the screen.

Entry Conditions

M1% = 2

Exit Conditions

None

Example

If the cursor flag value is 0, you can use these statements to hide the cursor:

```
100  '
200  '  Hide the cursor
300  '
400  M1% = 2
500  CALL MOUSE(M1%,M2%,M3%,M4%)
```

GetPos Function Call 3

Get Mouse Position and Button Status

Returns the status of the left and right buttons and the horizontal and vertical positions of the cursor.

The button status is a single integer value. Bits 0 and 1 represent the left and right buttons, respectively. A bit value is 0 if the button is released and 1 if it is pressed.

Entry Conditions

M1% = 3

Exit Conditions

M2% = *button status*
M2% = 0: button released
M2% = 1: button pressed
M3% = *horizontal cursor position*
M4% = *vertical cursor position*

The cursor positions are always within the range of minimum and maximum values of the virtual screen. (See "Virtual Screen" in Chapter 7.)

Example

```
100  '
200  '  Get current cursor positions, check
button status.
300  '
400  M1% = 3
500  CALL MOUSE(M1%,M2%,M3%,M4%)
600  IF M2% AND 1 THEN PRINT "Left button down."
700  IF M2% AND 2 THEN PRINT "Right button
down."
```

SetPos — Function Call 4

Set Mouse Cursor Position

Sets the cursor to the specified horizontal and vertical screen positions.

If the screen is not in high resolution mode, the values are rounded to the nearest horizontal or vertical values permitted for the current screen mode. (See "Virtual Screen" in Chapter 7.)

Entry Conditions

M1% = 4
M3% = *new horizontal cursor position*
M4% = *new vertical cursor position*

The new values must be within the horizontal and vertical ranges of the virtual screen.

Exit Conditions

None

Example

Assume that HMAX and VMAX contain the maximum horizontal and vertical positions values for the virtual screen. To set the cursor to the center of the screen, use these statements:

```
100  '
200  '  Put cursor in center of screen
300  '
400  M1% = 4
500  M3% = INT(HMAX/2)
600  M4% = INT(VMAX/2)
700  CALL MOUSE(M1%,M2%,M3%,M4%)
```

GetButtonPress — Function Call 5

Get Button Press Information

Returns the current button status, a count of button presses since the last call to this function, and the horizontal and vertical position of the cursor at the last press of the button.

Entry Conditions

M1% = 5
M2% = *button checked*
M2% = 0: left button
M2% = 1: right button

Exit Conditions

M1% = *button status*
M1% = 0: button released
M1% = 1: button pressed
M2% = *count of button presses*
M3% = *horizontal position at last press*
M4% = *vertical position at last press*

The *button status* is a single integer value. Bits 0 and 1 represent the left and right buttons, respectively. A bit value is 0 if the button is released and 1 if it is pressed.

The *count of button presses* is always in the range 0 to 32767; overflow is not detected. The count is set to 0 after the call.

The *horizontal* and *vertical* values are in the ranges defined by the virtual screen. Notice that these values represent the cursor position at the last press of the button, **not** the current cursor position.

Example

```
100  '
200  '  Get cursor position at last button
press.
300  '
400  M1% = 5
500  M2% = 0    ' left button
600  CALL MOUSE(M1%,M2%,M3%,M4%)
700  IF (M1% AND 1) THEN PRINT "Left button
down."
```

SetHorizontal — Function Call 7

Set Minimum and Maximum Horizontal Positions

Sets the minimum and maximum horizontal cursor positions on the screen. Subsequent cursor motion is restricted to the specified area. The minimum and maximum values are defined by the virtual screen. (See "Virtual Screen" in Chapter 7.)

If the cursor is outside the area when the call is made, it moves to just inside the area. If the minimum value is greater than the maximum, the two values are swapped.

Entry Conditions

M1% = 7
M3% = *minimum position*
M4% = *maximum position*

Exit Conditions

None

Example

```
100  '
200  '  Limit cursor to horizontal positions
below 150
300  '
400  M1% = 7
500  M3% = 0
600  M4% = 150
700  CALL MOUSE(M1%,M2%,M3%,M4%)
```

SetVertical — Function Call 8

Set Minimum and Maximum Vertical Positions

Sets the minimum and maximum vertical cursor positions on the screen. Subsequent cursor motion is restricted to the specified area. The minimum and maximum values are defined by the virtual screen. (See "Virtual Screen" in Chapter 7.)

If the cursor is outside the area when the call is made, it moves to just inside the area. If the minimum value is greater than the maximum, the two values are swapped.

Entry Conditions

M1% = 8
M3% = *minimum position*
M4% = *maximum position*

Exit Conditions

None

Example

```
100  '
200  '  Limit cursor to vertical positions
300  '  between 100 and 150
400  '
500  M1% = 8
600  M3% = 100
700  M4% = 150
800  CALL MOUSE(M1%,M2%,M3%,M4%)
```

SetCursorBlock Function Call 9

Set Graphics Cursor Block

Defines the shape, color, and center of the cursor when in graphics mode.

The function uses the values found in the screen mask and the cursor mask to build the cursor shape and color. (See Chapter 8, "Defining Graphics Cursors.")

To pass the screen mask and the cursor mask in BASIC, assign their values to an integer array and use the first element of the array as the fourth parameter in the call. (See the example.)

To pass the screen and cursor masks in assembly language, assign their values to 2 contiguous arrays and pass the address of the first array in register DX. Be sure to load the segment address of the arrays in the ES register before making the call.

The cursor hot spot values must define 1 pixel within the cursor. (See "Graphics Cursor Hot Spot" in Chapter 8.) The values must be in the range -16 to 16.

Entry Conditions

M1% = 9
M2% = *horizontal cursor hot spot*
M3% = *vertical cursor hot spot*
M4% = *pointer to screen and cursor masks*

Exit Conditions

None

Example

To define a cursor in high-resolution graphics mode, first define the values to the cursor array and then make the call:

```
100  '
200  '  Define the screen mask
300  '
400  CURSOR(0,0)=&HFFFF  '1111111111111111
500  CURSOR(1,0)=&HFFFF  '1111111111111111
600  CURSOR(2,0)=&HFFFF  '1111111111111111
700  CURSOR(3,0)=&HFFFF  '1111111111111111
800  CURSOR(4,0)=&HFFFF  '1111111111111111
900  CURSOR(5,0)=&HFFFF  '1111111111111111
1000 CURSOR(6,0)=&HFFFF  '1111111111111111
1100 CURSOR(7,0)=&HFFFF  '1111111111111111
1200 CURSOR(8,0)=&HFFFF  '1111111111111111
1300 CURSOR(9,0)=&HFFFF  '1111111111111111
1400 CURSOR(10,0)=&HFFFF '1111111111111111
1500 CURSOR(11,0)=&HFFFF '1111111111111111
1600 CURSOR(12,0)=&HFFFF '1111111111111111
1700 CURSOR(13,0)=&HFFFF '1111111111111111
1800 CURSOR(14,0)=&HFFFF '1111111111111111
1900 CURSOR(15,0)=&HFFFF '1111111111111111
2000 '
2100 '  Define the cursor mask
2200 '
2300 CURSOR(0,1)=&H8000  '1000000000000000
2400 CURSOR(1,1)=&HE000  '1110000000000000
2500 CURSOR(2,1)=&HF800  '1111100000000000
2600 CURSOR(3,1)=&HFE00  '1111111000000000
2700 CURSOR(4,1)=&HD800  '1101100000000000
2800 CURSOR(5,1)=&H0C00  '0000110000000000
2900 CURSOR(6,1)=&H0600  '0000011000000000
3000 CURSOR(7,1)=&H0300  '0000001100000000
3100 CURSOR(8,1)=&H0000  '0000000000000000
3200 CURSOR(9,1)=&H0000  '0000000000000000
3300 CURSOR(10,1)=&H0000 '0000000000000000
3400 CURSOR(11,1)=&H0000 '0000000000000000
3500 CURSOR(12,1)=&H0000 '0000000000000000
3600 CURSOR(13,1)=&H0000 '0000000000000000
3700 CURSOR(14,1)=&H0000 '0000000000000000
3800 CURSOR(15,1)=&H0000 '0000000000000000
3900 '
4000 '  Define cursor shape, color, and center
4100 '
4200 M1% = 9
4300 M2% = 0    ' Horizontal hot spot
4400 M3% = 0    ' Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

SetText — Function Call 10

Set Text Cursor

Selects the software or hardware text cursor. If you select the software text cursor, this function defines the character attributes of the cursor when in text mode. If you select the hardware text cursor, this function defines the first and last scan lines to be shown on the screen.

Entry Conditions

M1% = 10
M2% = *cursor type*
M2% = 0: software text cursor
M2% = 1: hardware text cursor
M3% = *screen mask value* (for software text cursor) or *number of first scan line* (for hardware text cursor)
M4% = *cursor mask value* (for software text cursor) or *number of last scan line* (for hardware text cursor)

The scan line numbers (M3% and M4%) for the hardware text cursor are in the range 0 to 7.

Exit Conditions

None

Example

To create a blinking 0 software text cursor, use these statements:

```
100  M1% = 10
110  M2% = 0          ' Select text cursor
120  M3% = &H7700     ' Screen mask
130  M4% = &H8830     ' Cursor mask
140  CALL MOUSE(M1%,M2%,M3%,M4%)
```

ReadCounters — Function Call 11

Read Mouse Motion Counters

Returns the horizontal and vertical mickey count since the last call to this function. The mickey count is the distance in 1/80 inch increments (0.32 mm increments) that the mouse has moved. (See "Mouse Unit of Distance: The Mickey" in Chapter 7.)

Entry Conditions

M1% = 11

Exit Conditions

M3% = *horizontal count*
M4% = *vertical count*

The mickey count is always within the range -32768 to 32767. A positive horizontal count specifies a motion to the right. A positive vertical count specifies a motion to the bottom of the screen. Overflow is ignored.

The mickey count is set to 0 after the call is completed.

Example

```
100  '
200  '  Get the mickey count
300  '
400  M1% = 11
500  CALL MOUSE(M1%,M2%,M3%,M4%)
```

SetMickeyPix — Function Call 15

Set Mickey/Pixel Ratio

Sets the mickey to pixel ratio for mouse motion. (See "Mouse Unit of Distance: The Mickey" in Chapter 7.) The horizontal and vertical ratios specify a number of mickeys per 8 pixels. The values must be in the range 1 to 32767.

Entry Conditions

M1% = 15
M3% = *horizontal mickey/pixel ratio*
Default = 8/8
M4% = *vertical mickey/pixel ratio*
Default = 16/8

In the default setting, 8.0 inches (20.3 cm) of mouse travel moves the cursor all the way across the screen, and 5.0 inches (12.7 cm) of travel moves it all the way down the screen.

Exit Conditions

None

Example

```
100  '
200  '  Set mickey/pixel ratio at 16 to 8 and 32
to 8
300  '
400  M1% = 15
500  M3% = 16          ' horizontal ratio
600  M4% = 32          ' vertical ratio
700  CALL MOUSE(M1%,M2%,M3%,M4%)
```

PIANO PROGRAM LISTING

This appendix presents the complete source to the PIANO Demonstration program. The program is written for the Tandy 1000's BASIC Interpreter. The following is an explanation of the program details:

Line Numbers	Comments
1000-1060	Copyright message.
1070-1160	Set up music, clear graphics screen to blue.
1170-1250	Read in the frequencies for the various piano keys.
1260-1380	Link the mouse software and the program.
1390-1430	Function 15 sets the mouse sensitivity. With this setting, a horizontal movement of 4 inches (10.2 cm) moves the cursor across the entire screen. This relatively high sensitivity permits songs to be played rapidly. Accuracy is no problem since the piano keys are large.
1440-1620	The integer array CURSOR contains the screen mask and the cursor mask. The masks define the shape and color of the cursor. These statements define the screen mask; the mask is set to all 1's. The mask will be logically ANDed with screen under the cursor.
1630-1810	These statements define the cursor mask. The values will be exclusively ORed with the result of the AND operation to create the cursor shape and color. In this case, the cursor shape is an upward-pointing arrowhead. Its color is different from whatever is below it.

1820-1860	Function 9 sets the cursor shape. It also defines the cursor hot spot. In this case, the hot spot is the tip of the arrowhead. The mouse software will automatically prevent the cursor hot spot from leaving the screen.
1870-1990	These statements initialize the keyboard size parameters.
2000-2150	These statements draw the "white" and "black" piano keys.
2160-2200	These statements draw the "quit" box in the lower right corner.
2210-2240	Function 4 centers the cursor to just under the piano keys.
2250	Function 1 turns on the cursor. The cursor appears on the screen and can be moved by using the mouse.
2260-2290	Function 3 gives the status of the 2 mouse buttons and the location of the cursor. This is probably the most common mouse function used in applications.
2300-2370	Some decision making is performed. If both mouse buttons are up, or if the mouse is not on the piano keyboard, then any sound that might be playing is turned off.
2380-2430	At this point, the mouse button is down over the quit box. The program turns off the mouse cursor, clears the screen, then quits.
2440-2510	The program has determined a button is down over the piano keyboard. These statements determine which key the mouse cursor is over.

2520-2570	The note is played by the SOUND statement set with the correct frequency. This note is played in the background as the program loops back to Line 2090.
2580-2630	This data contains the correct frequency to play the musical notes.

```
1000 '
1010 '                    THE VIRTUAL PIANO
1020 '
1030 '  COPYRIGHT (C) 1983 BY MICROSOFT CORPORATION
1040 '              WRITTEN BY CHRIS PETERS
1050 '
1060 '-----------------------------------------
1070 '
1080 '  INITIALIZE
1090 '
1100 DEFINT A-Z
1110 DIM CURSOR(15,1),FREQ(27,2)
1120 KEY OFF
1130 PLAY"MF"
1140 SC=1: SCREEN SC
1150 COLOR 1,1
1160 CLS
1170 '
1180 '  Read in the flat, normal, and sharp note
frequencies
1190 '
1200 FOR J=0 TO 2
1210 FOR I=0 TO 6
1220 READ K
1230 FREQ(I,J)=K : FREQ(I+7,J)=K*2 :
FREQ(I+14,J)=K*4 : FREQ(I+21,J)=K*8
1240 NEXT
1250 NEXT
1260 '
1270 '  Determine Mouse Driver location, if not
found, quit.
1280 '
1290 DEF SEG=0
1300 MSEG=256*PEEK(51*4+3)+PEEK(51*4+2)
' Get mouse segment
1310 MOUSE=256*PEEK(51*4+1)+PEEK(51*4)+2
' Get mouse offset
1320 IF MSEG AND MOUSE THEN 1370
1330 PRINT"Mouse driver not found"
' Not found so print error.
1340 PRINT
1350 PRINT"Press any key to return to system"
1360 I$=INKEY$ : IF I$="" THEN 1360 ELSE SYSTEM
1370 DEF SEG=MSEG                    ' Set mouse segment
1380 M1 = 0 : CALL MOUSE(M1,M2,M3,M4)
' Initialize the mouse
1390 '
1400 ' Set mouse sensitivity
1410 '
1420 M1=15 : M3=4 : M4=8
1430 CALL MOUSE(M1,M2,M3,M4)
1440 '
```

```
1450 '  Define the "logical and" cursor mask
1460 '
1470 CURSOR( 0,0)=&HFFFF    ' Binary 1111111111111111
1480 CURSOR( 1,0)=&HFFFF    ' Binary 1111111111111111
1490 CURSOR( 2,0)=&HFFFF    ' Binary 1111111111111111
1500 CURSOR( 3,0)=&HFFFF    ' Binary 1111111111111111
1510 CURSOR( 4,0)=&HFFFF    ' Binary 1111111111111111
1520 CURSOR( 5,0)=&HFFFF    ' Binary 1111111111111111
1530 CURSOR( 6,0)=&HFFFF    ' Binary 1111111111111111
1540 CURSOR( 7,0)=&HFFFF    ' Binary 1111111111111111
1550 CURSOR( 8,0)=&HFFFF    ' Binary 1111111111111111
1560 CURSOR( 9,0)=&HFFFF    ' Binary 1111111111111111
1670 CURSOR(10,0)=&HFFFF    ' Binary 1111111111111111
1580 CURSOR(11,0)=&HFFFF    ' Binary 1111111111111111
1590 CURSOR(12,0)=&HFFFF    ' Binary 1111111111111111
1600 CURSOR(13,0)=&HFFFF    ' Binary 1111111111111111
1610 CURSOR(14,0)=&HFFFF    ' Binary 1111111111111111
1620 CURSOR(15,0)=&HFFFF    ' Binary 1111111111111111
1630 '
1640 '  Define the "exclusive or" cursor mask
1650 '
1660 CURSOR( 0,1)=&H300     ' Binary 0000001100000000
1670 CURSOR( 1,1)=&H300     ' Binary 0000001100000000
1680 CURSOR( 2,1)=&HFC0     ' Binary 0000111111000000
1690 CURSOR( 3,1)=&HFC0     ' Binary 0000111111000000
1700 CURSOR( 4,1)=&H3FF0    ' Binary 0011111111110000
1710 CURSOR( 5,1)=&H3FF0    ' Binary 0011111111110000
1720 CURSOR( 6,1)=&HFCFC    ' Binary 1111110011111100
1730 CURSOR( 7,1)=&HC00C    ' Binary 1100000000001100
1740 CURSOR( 8,1)=&H0       ' Binary 0000000000000000
1750 CURSOR( 9,1)=&H0       ' Binary 0000000000000000
1760 CURSOR(10,1)=&H0       ' Binary 0000000000000000
1770 CURSOR(11,1)=&H0       ' Binary 0000000000000000
1780 CURSOR(12,1)=&H0       ' Binary 0000000000000000
1790 CURSOR(13,1)=&H0       ' Binary 0000000000000000
1800 CURSOR(14,1)=&H0       ' Binary 0000000000000000
1810 CURSOR(15,1)=&H0       ' Binary 0000000000000000
1820 '
1830 '  Set the mouse cursor shape
1840 '
1850 M1 = 9 : M2 = 6 : M3 = 0
1860 CALL MOUSE(M1,M2,M3,CURSOR(0,0))
1870 '
1940 '
1950 '  Initialize keyboard size parameters
1960 '
1970 YL = 60 : WKL = 80 : BKL=45 : KW = 15 : WKN = 21
1980 XL = 320-KW*WKN : YH = YL + WKL : XH = 319 :
BKW2=KW\3
1990 QX = 272 : QY = 176
2000 '
```

```
2010 '  Draw the "white" keys
2020 '
2030 LINE(XL,YL)-(XH,YH),3,BF
2040 FOR I=XL TO XH STEP KW
2050 LINE(I,YL)-(I,YH),0
2060 NEXT
2070 '
2080 '  Draw the "black" keys
2090 '
2100 C=6
2110 FOR X=XL TO XH STEP KW
2120 C=C+1 : IF C=7 THEN C=0
2130 IF C=0 OR C=3 THEN 2150
2140 LINE(X-BKW2,YL)-(X+BKW2,YL+BKL),2,BF
2150 NEXT
2160 '
2170 '  Draw the quit box
2180 '
2190 LINE(QX,QY)-(319,199),3,B
2200 LOCATE 24,36 : PRINT"Quit";
2210 '
2220 '  Set mouse cursor location, then turn on
     cursor
2230 '
2240 M1 = 4 : M3 = 320 : M4 = 160 : CALL
     MOUSE(M1,M2,M3,M4)
2250 M1 = 1: CALL MOUSE(M1,M2,M3,M4)
2260 '
2270 '  M A I N L O O P
2280 '
2290 M1 = 3 : CALL MOUSE(M1,BT,MX,MY)
     ' Get mouse location and button status
2300 IF (BT AND 2) THEN OTV=7: GOTO 2340
     ' If right button down, set high octave
2310 IF (BT AND 1) THEN OTV=0: GOTO 2340
     ' If left button down, set lower octave
2320 SOUND 442,0          ' If both buttons up, turn off
                            sound
2330 GOTO 2290            ' Keep looping...
2340 MX = MX\2            ' Correct for medium
                            resolution screen
2350 IF MX <= XL OR MY < YL THEN 2320
     ' If above keyboard, turn off sound
2360 IF MY <= YH THEN 2470
     ' If on keyboard, play sound
2370 IF MY < QY OR MX < QX THEN 2320
     ' If above quit box, turn off sound
2380 '
2390 '  Button down inside the quit box
2400 '
```

```
2410 M1=2: CALL MOUSE(M1,M2,M3,M4)
     ' Turn off mouse cursor
2420 CLS          ' Clear screen
2430 SYSTEM       ' Quit
2440 '
2450 '  Button down over keyboard, determine which
     key
2460 '
2470 WKY = (MX-XL)\KW+OTV :R = 1
     ' Get which "white" key cursor is over
2480 IF MY > YL+BKL THEN 2560
     ' Is it lower than the "black" keys?
2490 MK=(MX-XL) MOD KW
     ' No, get which side of key
2500 IF MK <= BKW2 THEN R+0 : GOTO 2560
     ' Is it the left "black" key?
2510 IF MK >= KW-BKW2 THEN R=2
     ' Is it the right "black" key?
2520 '
2530 '  Play the note. For BASIC interpreter
     duration = 2
2540 '                    For BASIC compiler
     duration = 1
2550 '
2560 SOUND FREQ(WKY,R),2
2570 GOTO 2290
     'Continue looping
2580 '
2590 '  Musical note frequencies
2600 '
2610 DATA 131,139,156,175,185,208,233
2620 DATA 131,147,165,175,196,220,247
2630 DATA 139,156,165,185,208,233,247
```

SAMPLE CURSORS

This appendix describes 8 sample graphics cursors. These sample cursors illustrate the wide variety of cursor shapes that you can define for use in BASIC application programs.

The sample cursors are designed for high-resolution graphics mode. Each cursor is a white shape with a black outline on a transparent field. The shape typically suggests the type of action you can take with the mouse. For example, an arrow usually means "make a selection by pointing at an item."

To use a sample cursor in your own BASIC program, copy the BASIC statements presented for the cursor directly to your program. Type the statements exactly as shown, using line numbers that are consistent with your program's numbering scheme.

To use a sample cursor in an assembly or high-level language program, define an array in your program and assign the values given for each cursor to the array elements. Assign the values in a way that makes their storage order identical to their storage order in a BASIC program.

The statements in this appendix define only the cursor's shape. It is up to you to define the action associated with the cursor by including the necessary statements in your program.

Standard Cursor

The standard cursor is a solid arrow that points up and to the left. The hot spot is directly beyond the arrow's tip; so you can point to an item without covering it. The standard cursor is the most convenient shape when using the mouse to choose items from the screen.

```
100'
200' Define the screen mask
300'
400 CURSOR( 0,0)=&H3FFF    'Binary 0011111111111111
500 CURSOR( 1,0)=&H1FFF    'Binary 0001111111111111
600 CURSOR( 2,0)=&H0FFF    'Binary 0000111111111111
700 CURSOR( 3,0)=&H07FF    'Binary 0000011111111111
800 CURSOR( 4,0)=&H03FF    'Binary 0000001111111111
900 CURSOR( 5,0)=&H01FF    'Binary 0000000111111111
1000 CURSOR( 6,0)=&H00FF   'Binary 0000000011111111
1100 CURSOR( 7,0)=&H007F   'Binary 0000000001111111
1200 CURSOR( 8,0)=&H003F   'Binary 0000000000111111
1300 CURSOR( 9,0)=&H001F   'Binary 0000000000011111
1400 CURSOR(10,0)=&H01FF   'Binary 0000000111111111
1500 CURSOR(11,0)=&H10FF   'Binary 0001000011111111
1600 CURSOR(12,0)=&H30FF   'Binary 0011000011111111
1700 CURSOR(13,0)=&HF87F   'Binary 1111100001111111
1800 CURSOR(14,0)=&HF87F   'Binary 1111100001111111
1900 CURSOR(15,0)=&HFC3F   'Binary 1111110000111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000   'Binary 0000000000000000
2400 CURSOR( 1,1)=&H4000   'Binary 0100000000000000
2500 CURSOR( 2,1)=&H6000   'Binary 0110000000000000
2600 CURSOR( 3,1)=&H7000   'Binary 0111000000000000
2700 CURSOR( 4,1)=&H7800   'Binary 0111100000000000
2800 CURSOR( 5,1)=&H7C00   'Binary 0111110000000000
2900 CURSOR( 6,1)=&H7E00   'Binary 0111111000000000
3000 CURSOR( 7,1)=&H7F00   'Binary 0111111100000000
3100 CURSOR( 8,1)=&H7F80   'Binary 0111111110000000
3200 CURSOR( 9,1)=&H78C0   'Binary 0111111111000000
3300 CURSOR(10,1)=&H7C00   'Binary 0111110000000000
3400 CURSOR(11,1)=&H4600   'Binary 0100011000000000
3500 CURSOR(12,1)=&H0600   'Binary 0000011000000000
3600 CURSOR(13,1)=&H0300   'Binary 0000001100000000
3700 CURSOR(14,1)=&H0300   'Binary 0000001100000000
3800 CURSOR(15,1)=&H0000   'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=-1 'Horizontal hot spot
4400 M3%=-1 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Upward Arrow

The upward-pointing arrow is a solid arrow with the hot spot at the tip. It is useful when directing a motion on the screen with the mouse.

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&HF9FF    'Binary 1111100111111111
 500 CURSOR( 1,0)=&HF0FF    'Binary 1111000011111111
 600 CURSOR( 2,0)=&HE07F    'Binary 1110000001111111
 700 CURSOR( 3,0)=&HE07F    'Binary 1110000001111111
 800 CURSOR( 4,0)=&HC03F    'Binary 1100000000111111
 900 CURSOR( 5,0)=&HC03F    'Binary 1100000000111111
1000 CURSOR( 6,0)=&H801F    'Binary 1000000000011111
1100 CURSOR( 7,0)=&H801F    'Binary 1000000000011111
1200 CURSOR( 8,0)=&H000F    'Binary 0000000000001111
1300 CURSOR( 9,0)=&H000F    'Binary 0000000000001111
1400 CURSOR(10,0)=&HF0FF    'Binary 1111000011111111
1500 CURSOR(11,0)=&HF0FF    'Binary 1111000011111111
1600 CURSOR(12,0)=&HF0FF    'Binary 1111000011111111
1700 CURSOR(13,0)=&HF0FF    'Binary 1111000011111111
1800 CURSOR(14,0)=&HF0FF    'Binary 1111000011111111
1900 CURSOR(15,0)=&HF0FF    'Binary 1111000011111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000    'Binary 0000000000000000
2400 CURSOR( 1,1)=&H0600    'Binary 0000011000000000
2500 CURSOR( 2,1)=&H0F00    'Binary 0000111100000000
2600 CURSOR( 3,1)=&H0F00    'Binary 0000111100000000
2700 CURSOR( 4,1)=&H1F80    'Binary 0001111110000000
2800 CURSOR( 5,1)=&H1F80    'Binary 0001111110000000
2900 CURSOR( 6,1)=&H3FC0    'Binary 0011111111000000
3000 CURSOR( 7,1)=&H3FC0    'Binary 0011111111000000
3100 CURSOR( 8,1)=&H7FE0    'Binary 0111111111100000
3200 CURSOR( 9,1)=&H0600    'Binary 0000011000000000
3300 CURSOR(10,1)=&H0600    'Binary 0000011000000000
3400 CURSOR(11,1)=&H0600    'Binary 0000011000000000
3500 CURSOR(12,1)=&H0600    'Binary 0000011000000000
3600 CURSOR(13,1)=&H0600    'Binary 0000011000000000
3700 CURSOR(14,1)=&H0600    'Binary 0000011000000000
3800 CURSOR(15,1)=&H0000    'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=5 'Horizontal hot spot
4400 M3%=0 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Left Arrow

The left-pointing arrow is a solid arrow with the hot spot at the tip. This shape is useful when directing a motion on the screen with the mouse. To generate a right arrow, reverse the binary bit pattern for each array element and move the hot spot to the new tip. For example, the first element, Binary 1111111000011111 (&HFE1F), becomes Binary 1111100001111111 (&HF87F).

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&HFE1F    'Binary 1111111000011111
 500 CURSOR( 1,0)=&HF01F    'Binary 1111000000011111
 600 CURSOR( 2,0)=&H0000    'Binary 0000000000000000
 700 CURSOR( 3,0)=&H0000    'Binary 0000000000000000
 800 CURSOR( 4,0)=&H0000    'Binary 0000000000000000
 900 CURSOR( 5,0)=&HF01F    'Binary 1111000000011111
1000 CURSOR( 6,0)=&HFE1F    'Binary 1111111000011111
1100 CURSOR( 7,0)=&HFFFF    'Binary 1111111111111111
1200 CURSOR( 8,0)=&HFFFF    'Binary 1111111111111111
1300 CURSOR( 9,0)=&HFFFF    'Binary 1111111111111111
1400 CURSOR(10,0)=&HFFFF    'Binary 1111111111111111
1500 CURSOR(11,0)=&HFFFF    'Binary 1111111111111111
1600 CURSOR(12,0)=&HFFFF    'Binary 1111111111111111
1700 CURSOR(13,0)=&HFFFF    'Binary 1111111111111111
1800 CURSOR(14,0)=&HFFFF    'Binary 1111111111111111
1900 CURSOR(15,0)=&HFFFF    'Binary 1111111111111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000    'Binary 0000000000000000
2400 CURSOR( 1,1)=&H00C0    'Binary 0000000011000000
2500 CURSOR( 2,1)=&H07C0    'Binary 0000011111000000
2600 CURSOR( 3,1)=&H7FFE    'Binary 0111111111111110
2700 CURSOR( 4,1)=&H07C0    'Binary 0000011111000000
2800 CURSOR( 5,1)=&H00C0    'Binary 0000000011000000
2900 CURSOR( 6,1)=&H0000    'Binary 0000000000000000
3000 CURSOR( 7,1)=&H0000    'Binary 0000000000000000
3100 CURSOR( 8,1)=&H0000    'Binary 0000000000000000
3200 CURSOR( 9,1)=&H0000    'Binary 0000000000000000
3300 CURSOR(10,1)=&H0000    'Binary 0000000000000000
3400 CURSOR(11,1)=&H0000    'Binary 0000000000000000
3500 CURSOR(12,1)=&H0000    'Binary 0000000000000000
3600 CURSOR(13,1)=&H0000    'Binary 0000000000000000
3700 CURSOR(14,1)=&H0000    'Binary 0000000000000000
3800 CURSOR(15,1)=&H0000    'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=0 'Horizontal hot spot
4400 M3%=3 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Checkmark

The checkmark is a solid figure with the hot spot in the center of the "V" formed by the check. It's useful when checking off items from a list with the mouse or while a program is checking some aspect of its operation.

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&HFFF0    'Binary 1111111111110000
 500 CURSOR( 1,0)=&HFFE0    'Binary 1111111111100000
 600 CURSOR( 2,0)=&HFFC0    'Binary 1111111111000000
 700 CURSOR( 3,0)=&HFF81    'Binary 1111111110000001
 800 CURSOR( 4,0)=&HFF03    'Binary 1111111100000011
 900 CURSOR( 5,0)=&H0607    'Binary 0000011000000111
1000 CURSOR( 6,0)=&H000F    'Binary 0000000000001111
1100 CURSOR( 7,0)=&H001F    'Binary 0000000000011111
1200 CURSOR( 8,0)=&HC03F    'Binary 1100000000111111
1300 CURSOR( 9,0)=&HF07F    'Binary 1111000001111111
1400 CURSOR(10,0)=&HFFFF    'Binary 1111111111111111
1500 CURSOR(11,0)=&HFFFF    'Binary 1111111111111111
1600 CURSOR(12,0)=&HFFFF    'Binary 1111111111111111
1700 CURSOR(13,0)=&HFFFF    'Binary 1111111111111111
1800 CURSOR(14,0)=&HFFFF    'Binary 1111111111111111
1900 CURSOR(15,0)=&HFFFF    'Binary 1111111111111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000    'Binary 0000000000000000
2400 CURSOR( 1,1)=&H0006    'Binary 0000000000000110
2500 CURSOR( 2,1)=&H000C    'Binary 0000000000001100
2600 CURSOR( 3,1)=&H0018    'Binary 0000000000011000
2700 CURSOR( 4,1)=&H0030    'Binary 0000000000110000
2800 CURSOR( 5,1)=&H0060    'Binary 0000000001100000
2900 CURSOR( 6,1)=&H70C0    'Binary 0111000011000000
3000 CURSOR( 7,1)=&H1D80    'Binary 0001110110000000
3100 CURSOR( 8,1)=&H0700    'Binary 0000011100000000
3200 CURSOR( 9,1)=&H0000    'Binary 0000000000000000
3300 CURSOR(10,1)=&H0000    'Binary 0000000000000000
3400 CURSOR(11,1)=&H0000    'Binary 0000000000000000
3500 CURSOR(12,1)=&H0000    'Binary 0000000000000000
3600 CURSOR(13,1)=&H0000    'Binary 0000000000000000
3700 CURSOR(14,1)=&H0000    'Binary 0000000000000000
3800 CURSOR(15,1)=&H0000    'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=6 'Horizontal hot spot
4400 M3%=7 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Pointing Hand

The upward-pointing hand is a solid figure with the hot spot at the tip of the extended finger. The pointing hand is another convenient shape to use when selecting items from the screen, especially if the items are represented by symbols such as the keys of a piano.

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&HE1FF    'Binary 1110000111111111
 500 CURSOR( 1,0)=&HE1FF    'Binary 1110000111111111
 600 CURSOR( 2,0)=&HE1FF    'Binary 1110000111111111
 700 CURSOR( 3,0)=&HE1FF    'Binary 1110000111111111
 800 CURSOR( 4,0)=&HE1FF    'Binary 1110000111111111
 900 CURSOR( 5,0)=&HE000    'Binary 1110000000000000
1000 CURSOR( 6,0)=&HE000    'Binary 1110000000000000
1100 CURSOR( 7,0)=&HE000    'Binary 1110000000000000
1200 CURSOR( 8,0)=&H0000    'Binary 0000000000000000
1300 CURSOR( 9,0)=&H0000    'Binary 0000000000000000
1400 CURSOR(10,0)=&H0000    'Binary 0000000000000000
1500 CURSOR(11,0)=&H0000    'Binary 0000000000000000
1600 CURSOR(12,0)=&H0000    'Binary 0000000000000000
1700 CURSOR(13,0)=&H0000    'Binary 0000000000000000
1800 CURSOR(14,0)=&H0000    'Binary 0000000000000000
1900 CURSOR(15,0)=&H0000    'Binary 0000000000000000
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H1E00    'Binary 0001111000000000
2400 CURSOR( 1,1)=&H1200    'Binary 0001001000000000
2500 CURSOR( 2,1)=&H1200    'Binary 0001001000000000
2600 CURSOR( 3,1)=&H1200    'Binary 0001001000000000
2700 CURSOR( 4,1)=&H1200    'Binary 0001001000000000
2800 CURSOR( 5,1)=&H13FF    'Binary 0001001111111111
2900 CURSOR( 6,1)=&H1249    'Binary 0001001001001001
3000 CURSOR( 7,1)=&H1249    'Binary 0001001001001001
3100 CURSOR( 8,1)=&HF249    'Binary 1111001001001001
3200 CURSOR( 9,1)=&H9001    'Binary 1001000000000001
3300 CURSOR(10,1)=&H9001    'Binary 1001000000000001
3400 CURSOR(11,1)=&H9001    'Binary 1001000000000001
3500 CURSOR(12,1)=&H8001    'Binary 1000000000000001
3600 CURSOR(13,1)=&H8001    'Binary 1000000000000001
3700 CURSOR(14,1)=&H8001    'Binary 1000000000000001
3800 CURSOR(15,1)=&HFFFF      'Binary
1111111111111111
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=5 'Horizontal hot spot
4400 M3%=0 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Diagonal Cross

The diagonal cross is a solid figure with the hot spot at the center of the cross. This shape is useful as a pointer in a game or when canceling an operation or deleting an item from a list.

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&H07E0    'Binary 0000011111100000
 500 CURSOR( 1,0)=&H0180    'Binary 0000000110000000
 600 CURSOR( 2,0)=&H0000    'Binary 0000000000000000
 700 CURSOR( 3,0)=&HC003    'Binary 1100000000000011
 800 CURSOR( 4,0)=&HF00F    'Binary 1111000000001111
 900 CURSOR( 5,0)=&HC003    'Binary 1100000000000011
1000 CURSOR( 6,0)=&H0000    'Binary 0000000000000000
1100 CURSOR( 7,0)=&H0180    'Binary 0000000110000000
1200 CURSOR( 8,0)=&H07E0    'Binary 0000011111100000
1300 CURSOR( 9,0)=&HFFFF    'Binary 1111111111111111
1400 CURSOR(10,0)=&HFFFF    'Binary 1111111111111111
1500 CURSOR(11,0)=&HFFFF    'Binary 1111111111111111
1600 CURSOR(12,0)=&HFFFF    'Binary 1111111111111111
1700 CURSOR(13,0)=&HFFFF    'Binary 1111111111111111
1800 CURSOR(14,0)=&HFFFF    'Binary 1111111111111111
1900 CURSOR(15,0)=&HFFFF    'Binary 1111111111111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000    'Binary 0000000000000000
2400 CURSOR( 1,1)=&H700E    'Binary 0111000000001110
2500 CURSOR( 2,1)=&H1C38    'Binary 0001110000111000
2600 CURSOR( 3,1)=&H0660    'Binary 0000011001100000
2700 CURSOR( 4,1)=&H1C38    'Binary 0000001111000000
2800 CURSOR( 5,1)=&H700E    'Binary 0000011001100000
2900 CURSOR( 6,1)=&H0000    'Binary 0001110000111000
3000 CURSOR( 7,1)=&H0000    'Binary 0111000000011100
3100 CURSOR( 8,1)=&H0000    'Binary 0000000000000000
3200 CURSOR( 9,1)=&H0000    'Binary 0000000000000000
3300 CURSOR(10,1)=&H0000    'Binary 0000000000000000
3400 CURSOR(11,1)=&H0000    'Binary 0000000000000000
3500 CURSOR(12,1)=&H0000    'Binary 0000000000000000
3600 CURSOR(13,1)=&H0000    'Binary 0000000000000000
3700 CURSOR(14,1)=&H0000    'Binary 0000000000000000
3800 CURSOR(15,1)=&H0000    'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=7 'Horizontal hot spot
4400 M3%=4 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Rectangular Cross

The rectangular cross is a solid figure with the hot spot at the center of the cross. The shape is useful as a pointer in a game or when inserting items into a list.

```
100'
200' Define the screen mask
300'
400 CURSOR( 0,0)=&HFC3F    'Binary 1111110000111111
500 CURSOR( 1,0)=&HFC3F    'Binary 1111110000111111
600 CURSOR( 2,0)=&HFC3F    'Binary 1111110000111111
700 CURSOR( 3,0)=&H0000    'Binary 0000000000000000
800 CURSOR( 4,0)=&H0000    'Binary 0000000000000000
900 CURSOR( 5,0)=&H0000    'Binary 0000000000000000
1000 CURSOR( 6,0)=&HFC3F   'Binary 1111110000111111
1100 CURSOR( 7,0)=&HFC3F   'Binary 1111110000111111
1200 CURSOR( 8,0)=&HFC3F   'Binary 1111110000111111
1300 CURSOR( 9,0)=&HFFFF   'Binary 1111111111111111
1400 CURSOR(10,0)=&HFFFF   'Binary 1111111111111111
1500 CURSOR(11,0)=&HFFFF   'Binary 1111111111111111
1600 CURSOR(12,0)=&HFFFF   'Binary 1111111111111111
1700 CURSOR(13,0)=&HFFFF   'Binary 1111111111111111
1800 CURSOR(14,0)=&HFFFF   'Binary 1111111111111111
1900 CURSOR(15,0)=&HFFFF   'Binary 1111111111111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000   'Binary 0000000000000000
2400 CURSOR( 1,1)=&H0180   'Binary 0000000110000000
2500 CURSOR( 2,1)=&H0180   'Binary 0000000110000000
2600 CURSOR( 3,1)=&H0180   'Binary 0000000110000000
2700 CURSOR( 4,1)=&H7FFE   'Binary 0111111111111110
2800 CURSOR( 5,1)=&H0180   'Binary 0000000110000000
2900 CURSOR( 6,1)=&H0180   'Binary 0000000110000000
3000 CURSOR( 7,1)=&H0180   'Binary 0000000110000000
3100 CURSOR( 8,1)=&H0000   'Binary 0000000000000000
3200 CURSOR( 9,1)=&H0000   'Binary 0000000000000000
3300 CURSOR(10,1)=&H0000   'Binary 0000000000000000
3400 CURSOR(11,1)=&H0000   'Binary 0000000000000000
3500 CURSOR(12,1)=&H0000   'Binary 0000000000000000
3600 CURSOR(13,1)=&H0000   'Binary 0000000000000000
3700 CURSOR(14,1)=&H0000   'Binary 0000000000000000
3800 CURSOR(15,1)=&H0000   'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=7 'Horizontal hot spot
4400 M3%=4 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

Hourglass

The hourglass is a solid figure with the hot spot at the center of the glass. You can use this shape to show that the operation in progress takes some time to complete.

```
 100'
 200' Define the screen mask
 300'
 400 CURSOR( 0,0)=&H0000    'Binary 0000000000000000
 500 CURSOR( 1,0)=&H0000    'Binary 0000000000000000
 600 CURSOR( 2,0)=&H0000    'Binary 0000000000000000
 700 CURSOR( 3,0)=&H0000    'Binary 0000000000000000
 800 CURSOR( 4,0)=&H8001    'Binary 1000000000000001
 900 CURSOR( 5,0)=&HC003    'Binary 1100000000000011
1000 CURSOR( 6,0)=&HE007    'Binary 1110000000000111
1100 CURSOR( 7,0)=&HF00F    'Binary 1111000000001111
1200 CURSOR( 8,0)=&HE007    'Binary 1110000000000111
1300 CURSOR( 9,0)=&HC003    'Binary 1100000000000011
1400 CURSOR(10,0)=&H8001    'Binary 1000000000000001
1500 CURSOR(11,0)=&H0000    'Binary 0000000000000000
1600 CURSOR(12,0)=&H0000    'Binary 0000000000000000
1700 CURSOR(13,0)=&H0000    'Binary 0000000000000000
1800 CURSOR(14,0)=&H0000    'Binary 0000000000000000
1900 CURSOR(15,0)=&HFFFF    'Binary 1111111111111111
2000'
2100' Define cursor mask
2200'
2300 CURSOR( 0,1)=&H0000    'Binary 0000000000000000
2400 CURSOR( 1,1)=&H7FFE    'Binary 0111111111111110
2500 CURSOR( 2,1)=&H6006    'Binary 0110000000000110
2600 CURSOR( 3,1)=&H300C    'Binary 0011000000001100
2700 CURSOR( 4,1)=&H1818    'Binary 0001100000011000
2800 CURSOR( 5,1)=&H0C30    'Binary 0000110000110000
2900 CURSOR( 6,1)=&H0660    'Binary 0000011001100000
3000 CURSOR( 7,1)=&H03C0    'Binary 0000001111000000
3100 CURSOR( 8,1)=&H0660    'Binary 0000011001100000
3200 CURSOR( 9,1)=&H0C30    'Binary 0000110000110000
3300 CURSOR(10,1)=&H1998    'Binary 0001100110011000
3400 CURSOR(11,1)=&H33CC    'Binary 0011001111001100
3500 CURSOR(12,1)=&H67E6    'Binary 0110011111100110
3600 CURSOR(13,1)=&H7FFE    'Binary 0111111111111110
3700 CURSOR(14,1)=&H0000    'Binary 0000000000000000
3800 CURSOR(15,1)=&H0000    'Binary 0000000000000000
3900'
4000' Define cursor shape, color, and hot spot
4100'
4200 M1%=9
4300 M2%=7 'Horizontal hot spot
4400 M3%=7 'Vertical hot spot
4500 CALL MOUSE(M1%,M2%,M3%,CURSOR(0,0))
```

INDEX

AND and XOR results 38-40
  black and-white mode 44
  4-color modes 48
  16-color modes 50
application programs
  incorporating the mouse 25-33
  secondary communications channel 21
assembly-language programs, mouse system calls from 32-33
AUTOEXEC.BAT 13
  adding to 19-20
  creating 19

BASIC programs, mouse system calls from 31-32
BASIC screen modes *see* screen modes
battery 7-8, 11
binary equivalents of colors
  black-and-white mode 43, 44
  4-color modes 48
  16-color modes 50
bits per pixel 27
black-and-white graphics cursors 43-46
button 29
  get press information (Function 5) 62-63
  get release information (Function 6) 64-65
  get status (Function 3) 60

CLOCKGET.EXE 13
  copying 15, 16
CLOCKSET.EXE 13
  copying 15, 16
clock, setting 20
color, relationship to bits per pixel 27, 35
colors, numbers and binary equivalents
  black-and-white mode 35
  4-color mode 35
  16-color mode 36
CONFIG.SYS 13
  adding to 17-18
  creating 17

cursor 28-29 *see also* graphics cursor, hardware text cursor, software text cursor
  rate of movement 29-30
  set mouse cursor position (Function 4) 61
cursor blocks, samples 37-38
cursor mask
  graphics cursor 38-42, 45, 48, 50
  text cursor 51, 52, 53

date and time 20
DIGI-Mouse
  anatomy 21
  buttons 29
  connecting 11
  mickey 29-30
  moving 22
  sensitivity 30
  surface requirements 21-22
DIGI-Mouse Controller/Calendar Utilities Diskette 13, 15, 16
dummy variable names 55

FlagDec 59
FlagInc 58
floppy diskette procedure for copying the software 15
function calls
  0 (GetMouseStat/Reset) 57
  1 (FlagInc) 58
  2 (FlagDec) 59
  3 (GetPos) 60
  4 (SetPos) 61
  5 (GetButtonPress) 62-63
  6 (GetButtonRelease) 64-65
  7 (SetHorizontal) 66
  8 (SetVertical) 67
  9 (SetCursorBlock) 68-69
  10 (SetText) 70
  11 (ReadCounters) 71
  12 (SetCallMask) 72-73
  15 (SetMickeyPix) 74

graphics cursor block, setting (Function 9) 68-69
  hot spot 42
  sample cursors *see* sample graphics cursors
GetButtonPress 62-63
GetButtonRelease 64-65
GetMouseStat/Reset 57
GetPos 60

hard disk procedure for transferring the software 16
hardware text cursor 28, 53
high-level language programs, mouse calls from 33
horizontal position, setting (Function 7) 66
hot spot 42

initializing the mouse 17-18
initializing the system clock 19-20
input mask, set (Function 12) 72-73
installing Mouse Controller/Calendar Board 7-11
interface 25-30
  buttons 29
  cursors 28-29
  internal cursor flag 30
  mickey 29-30
  screen modes 25, 27-28
  virtual screen 26-27
internal cursor flag 30
  decrementing 30, 59
  incrementing 30, 58
  resetting 30, 57
IR3 21

mask bit values
  black-and-white mode 44
  4-color modes 40, 48
  16-color modes 50
mickey 29-30
mickey/pixel ratio, set (Function 15) 74

mouse
  anatomy 21
  cursor position, set (Function 4) 61
  functions *see* function calls
  interface *see* interface
  marking the range of an action 24
  motion counters, read (Function 11) 71
  moving 22
  selecting options with 24
  sensitivity 30
  surface requirements 21-22
  system calls 31-33 *see also* function calls
  using with PIANO.BAS 22-24
Mouse Controller/Calendar Board, installing 7-11
mouse driver 13
  copying 15, 16
  memory requirements 18
MOUSE.SYS 13
  copying 15, 16

PIANO.BAS 33
  copying 15, 16
  exiting 24
  explanation of program details 75-77
  program listing 78-81
  running 22-24
programming for mouse 25-33

ReadCounters 71

sample graphics cursors
  black-and-white cross 43-46
  checkmark 87
  diagonal cross 89
  hourglass 91
  left arrow 86
  pointing hand 88
  rectangular cross 90
  standard 84
  upward arrow 85

screen mask
  graphics cursor 38-42, 45, 48, 50
  software text cursor 51, 52, 53
screen mode attributes 27
screen mode, legal virtual coordinates 27-28
SetCallMask 31, 72-73
SetCursorBlock 68-69
SetHorizontal 66
SetMickeyPix 74
SetText 70
SetVertical 67
software 13
  copying to a system diskette 15
  transferring to hard disk 16
software text cursor 28, 51-53
static charge buildup 7, 9
system calls 55-74 *see also* function calls
  from assembly-language programs 32-33
  from BASIC interpreter 31-32
  from high-level languages 33
system cursor *see* text cursor

text cursor 28
  hardware 53
  software 51-53
text modes 28

vertical position, setting (Function 8) 67
virtual coordinates for various modes 27-28
virtual screen 26-28

XOR and AND results 39, 40

YEAR.DAT 20

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U. K.
91 KURRAJONG AVENUE MOUNT DRUITT, N.S.W. 2770	PARC INDUSTRIEL 5140 NANINNE (NAMUR)	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

8749560-2/85-BCo

Printed in U.S.A.